ARMY RESEARCH LABORATORY

A Demonstration to Assess Effectiveness, Suitability, and Survivability With the Missions and Means Framework

by Beth S. Ward, Paul J. Tanenbaum, Keon U. Burley, Paul H. Deitz, Britt E. Bray, Richard S. Sandmeyer, and Jack H. Sheehan

ARL-TR-6271 **December 2012**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5068

ARL-TR-6271 **December 2012**

A Demonstration to Assess Effectiveness, Suitability, and Survivability With the Missions and Means Framework

Beth S. Ward, Paul J. Tanenbaum, and Keon U. Burley
Survivability/Lethality Analysis Directorate, ARL

Paul H. Deitz
U.S. Army Materiel Systems Analysis Activity

Britt E. Bray
Dynamics Research Corporation

Richard S. Sandmeyer and Jack H. Sheehan
ORSA Corporation

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
December 2012	Final	May 2004–April 2005

4. TITLE AND SUBTITLE
A Demonstration to Assess Effectiveness, Suitability, and Survivability With the Missions and Means Framework

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Beth S. Ward, Paul J. Tanenbaum, Keon U. Burley, Paul H. Deitz,[*] Britt E. Bray,[†] Richard S. Sandmeyer,[‡] and Jack H. Sheehan[‡]

5d. PROJECT NUMBER
W911QX-09-F-0116

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-SLB-A
Aberdeen Proving Ground, MD 21005-5068

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6271

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
U.S. Army Materiel Systems Analysis Activity,[*] Dynamics Research Corporation,[†] ORSA Corporation[‡]

14. ABSTRACT

We describe a proof-of-principle exercise in which we demonstrated the use of a methodology called the Missions and Means Framework (MMF) to explore the behavior of a force in a notional operational context. The MMF is a methodology to specify military missions and evaluate alternative services and products in doctrine, organization, training, materiel, leader development, personnel, and facilities for their utility to those missions. The objective of the exercise was to explore the MMF's utility for (1) modeling a company's ability to continue its mission as a networked system of systems while suffering loss of capability in selected systems and (2) tracing the impact on critical mission tasks of degradation in system functions. The simulation showed how mission accomplishment could be modeled as a function of changes in the state of low-level components. The demonstration verified that the MMF is a useful framework to evaluate the effectiveness, suitability, and survivability of complex warfighting systems in the context of their contribution to the operational mission. We also present lessons learned on how the methods can be applied to the Army Test and Evaluation Command's strategy of mission-based test and evaluation.

15. SUBJECT TERMS
Missions and Means Framework (MMF), MBT&E, TOEL, $O_{3,4}$, criticality analysis, DOTMLPF, JCIDS, LFT&E

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Beth Ward
Unclassified	Unclassified	Unclassified	SAR	170	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6315

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Preface

The intent of this report is to provide general information on an improved methodology for assessing military systems and systems of systems. All quantitative results provided are purely notional and are included for illustrative purposes only. They are not reflective of the vulnerability or lethality of any specific vehicle, munition, configuration, or the like.

Acknowledgments

The authors thank Janet Shindell (U.S. Army Research Laboratory) and Chris Cosgrove (SURVICE Engineering) for their technical expertise and assistance in the criticality analysis of the platforms used for this demonstration.

INTENTIONALLY LEFT BLANK.

Executive Summary

Organizations within the U.S. Department of Defense have been collaborating for several years to develop a new methodology for specifying military missions and quantitatively evaluating alternative doctrine, organization, training, materiel, leader development, personnel, and facilities (DOTMLPF) services and products for their utility to those missions. This methodology, called the Missions and Means Framework (MMF), provides a structured way to describe key elements of military operations in a systematic procedure that explicitly specifies the mission and assesses mission accomplishment.

The procedure begins with mission decomposition—a process of specifying the tasks necessary to accomplish the mission. Mission decomposition is scalable across the levels of war: it can begin at the strategic national level with the national and theater (SN and ST) tasks (from the Universal Joint Task List (UJTL)) required to achieve national or alliance security objectives and guidance, and can extend through the operational level down to whatever level of operations is necessary to address the analytical objectives and associated questions. The lowest level of decomposition can include standard tasks to be performed by small units, platforms, or individual Soldiers. Each task is then assessed in the context of a doctrinal or specified operational mission to determine the capabilities and functional requirements necessary to accomplish the task's purpose. The analysis considers conditions and standards imposed by either a generic or scenario-specific operational environment. The resulting set of required capabilities and functions is then correlated to the materiel and personnel (supported by the relevant doctrine, organization, training, leadership, and facilities solutions) available to complete each task. Requirements that cannot be met by available materiel, personnel, and organizational resources constitute capability gaps.

In May 2004, the utility of the MMF methodology was demonstrated by the U.S. Army Research Laboratory's Survivability/Lethality Analysis Directorate and Dynamics Research Corporation at the direction of the Deputy Under Secretary of the Army for Operations Research. The demonstration, a simulation-driven project, showed that the MMF was effective, suitable, and survivable for application. In the simulation, the mission context featured a Mounted Combat System (MCS) company that was equipped with the Future Combat System and engaging a notional enemy force in a future joint tactical environment. The two key objectives for demonstrating the utility of the MMF were (1) to model the company's ability to continue its mission as a networked system of systems even as it suffered degradation in its systems' capabilities and (2) to trace the impact of those functional degradations on critical mission tasks.

The MMF provided the structure to capture and organize the data: mission decomposition, criticality analysis, and the correlation of mission tasks to required system functions. We analyzed a tactical mission by applying the principles of Army doctrinal processes for planning

and conducting operations. This produced a set of tasks (implied or essential) derived from existing authoritative task lists. A criticality analysis of the unit's platforms identified the systems' functions and component fault trees. Then a small group of operational experts correlated the tasks to required system functions. To evaluate the mission, we used select government and commercial off-the-shelf tools to incorporate the tasks and required system functional fault trees into a simulation model. Because the simulation focused on the main characters of the vignette rather than other details typical of force-level models, we named it the *Storyboard Model*. It demonstrated how mission accomplishment could be modeled as a function of changes in the state of low-level components. We showed how the component fault trees used to describe system functionality can support correlating the required functions to task execution—thus avoiding use of averaged utility values. Modeling this task-to-function association demonstrated a method to assess how the MCS company mission was affected by damage to or deterioration of components.

Through the Storyboard Model, the demonstration verified that the MMF is a useful framework to evaluate the effectiveness, suitability, and survivability of complex warfighting systems in the context of their contribution to the operational mission. Our purpose here is to present lessons learned and thoughts on how the methods demonstrated can be applied to the U.S. Army Test and Evaluation Command's strategy of mission-based test and evaluation.

1. Introduction

The Army is undergoing a transformation both in its doctrine for command and control and in its warfighting technologies. This transformation is rooted in the concept of decentralized decision making enabled by advanced network technologies. Thus, the future force is intended to be a collaborative, adaptive system of systems (SoS) able to quickly dominate the threat across the spectrum of conflict.

But traditional Army evaluations have been based on technical performance requirements whose suitability is strained by this transition [*1–4*]. Further, the vulnerability and lethality (V/L) data in the evaluation process have generally been misused. Historically, V/L data are calculated to express the loss of such high-level platform capabilities as "mobility or firepower" (M/F), which is then weighted by the capability's overall importance in some universal aggregation of missions. Army models and simulations then apply these universal results to particular missions in particular scenarios [*4*].

Organizations within the U.S. Department of Defense have been collaborating for several years to develop the Missions and Means Framework (MMF) [*1, 5*] to quantitatively evaluate alternative doctrine, organization, training, materiel, leader development, personnel, and facilities (DOTMLPF) services and products (see appendix A for details on the MMF). As part of that evaluation, the MMF is a structured approach for describing key elements of military operations in a disciplined, repeatable way to assess mission accomplishment. The MMF is particularly useful in the military decision-making process (MDMP) [*6*] to fully characterize a mission and can tightly link it to a high-resolution representation of people and materiel or at any intermediate level. The MMF aligns system components and functions to a specified tactical mission at a finer resolution than, say, mobility or firepower. The approach then evaluates system capability requirements of a mission in addition to technical performance parameters.

To capture such finer points and the sensitivity to mission, the complete MMF description includes not only the allied (blue) forces, but also the opposing (red) forces. Thus, when completely instantiated, the MMF view of the world includes each side's mission and their supporting people and materiel, as well as the interactions that occur between the sides.

The following sections describe an incremental change in the way of mounting military simulations. They also demonstrate war-game features that are new in both completeness and resolution.

1.1 Problems and Limitations Under Examination

The new methods illustrated in this report have their roots in the U.S. Army ballistic live-fire work of the mid 1980s and the subsequent developments [*2, 3*]. During the 1980s, it became

clear to both the Department of the Army and the Director, Operational Test and Evaluation (DOT&E) that the vulnerability models of the time were seriously deficient for use to support, replicate, and potentially replace actual, live field shots. Addressing the problems and limitations is a key motivation for the MMF.

Possibly the most important issue is that the so-called "probability of kill" (P_K) metrics in use [*4*] are highly problematic. The way the P_K metrics have always been generated mixes and confounds three notions that must be understood and handled distinctly:

1. The state, or condition, of the people and materiel under consideration (this can be thought of as an enumeration of what in the platform is working and what is not working).
2. The corresponding capability of the people and materiel.
3. Their utility (or effectiveness) for the intended mission purpose.

As a result, P_K's cannot be directly compared to any real-world observable [*3*, *4*] and hence cannot be empirically validated. Furthermore, because they have always been called probabilities and expressed in the unit interval (which is to say subject to $0 \leq P_K \leq 1.0$), war games have for many decades used them—out of convenience but without theoretic justification—as though they actually were probabilities. A final critical drawback to P_K's is that the vulnerability analysts who generate them cannot know the mission requirements that will be appropriate for any particular future run of a yet-unspecified combat simulation. So it should not fall to the vulnerability analysts to perform the step from functionality to effectiveness; this is properly left to the Soldier, operator, or combat modeler [*7*].

Even if the P_K's were indeed probabilities, they would be able to distinguish only two outcomes: killed or not killed (they treat the phenomena as Bernoulli trials, here at the platform level). It is true that binary outcomes may be appropriate for extreme ballistic interactions involving either significant overmatch or negligible damage. But most platform dysfunction that is of interest to the technical and operational communities corresponds to intermediate accumulations of damage, perhaps none of which by itself prevents a platform from pursuing its mission but which, in the aggregate, does thwart the platform's mission.

Besides resolving those shortcomings of traditional approaches, the MMF also provides the following benefits.

1.1.1 Linkage of Discrete Damage to Corresponding Capability

Over the years, the Ballistic Research Laboratory and later the Army Research Laboratory (ARL), had developed enhanced methodologies [*2*, *5*] for stochastically estimating which vehicle components would likely be damaged by a specific ballistic interaction and then estimating the resulting change in the vehicle's ability to move, shoot, communicate, and so forth. These damage and capability states are observable. Using the enhanced methodologies for modeling

damage interaction and its effect on performance, we can estimate vehicle damage and any corresponding change in a capability state. Multiple events or interactions, such as ballistic and reliability failure, can be modeled to estimate the combined damage as well as the effects on capabilities.

1.1.2 Time-Resolved Integration of Damage/Capability States

The MMF's mapping between physical state and capability is dependent on what causes the physical state to change; it permits combining any such changes regardless of mechanism (e.g., ballistic damage, reliability, failure, or electronic interference). This makes it possible to estimate the running, cumulative state of platforms as they proceed through the vicissitudes of any mission. This stands in contrast to the approach of most war games, for which a primary focus for many decades has been the overall loss-exchange (or killer/victim) ratio (LER). Though of continuing utility, LERs are inadequate to express today's complex mission goals.

1.1.3 Mission-Based Effectiveness Metrics

Over the past two decades, the warfighting community (joint and service-level) has developed formal semantics for describing missions in terms of a paradigm incorporating tasks, conditions, and standards (T/C/S). We adopt this same approach, formulating mission goals at multiple levels of war. And the conditions and standards implicitly express performance requirements for people and materiel, thus providing the proper mission context for assessing their effectiveness.

1.1.4 System-of-System (SoS) Evaluation in Proper Context

Over the past two decades, the issue of SoS performance has become very important. Just as redundant components on an individual platform can add to robustness, multiple platforms on the battlefield can withstand dysfunction or seize opportunity by trading off performance roles. However, the benefits and drawbacks of any SoS can only be assessed in the context of the operator's tasks, conditions, and standards. In the operational test community, mission-based testing contexts [*8–10*] are making their way into developmental and operational test strategies. Not only is individual platform/entity performance described by T/C/S, so too are *aggregations* of platforms at appropriate levels of multiplicity. The warfighter calls these objectives *collective tasks*, and we believe characterizing the collective tasks is the key to assessing both the benefits and burdens of SoS performance and effectiveness, i.e., the efficacy of an SoS. Using a football metaphor, we can describe the collective tasks as the equivalent of the team playbook—it is about how the performance of each team member on an assigned task contributes to the team endeavor.

1.2 Levels and Operators

With a goal to provide a formal description of the MDMP, the MMF organizes and specifies military operations in terms of 11 fundamental constructs for reasoning about the problems, i.e., elemental levels of content and transformations. Content is organized into the following seven levels:

Level 7: Purpose and Mission

Level 6: Context and Environment

Level 5: Index and Location/Time

Level 4: Tasks and Operations

Level 3: Functions and Capabilities

Level 2: Components and Forces

Level 1: Interactions and Effects

In addition, the following four operators are included to express the relationships between levels:

$O_{1,2}$: level-1 interaction specifications into level-2 component states

$O_{2,3}$: level-2 component states into level-3 functional performance

$O_{3,4}$: level-3 functional performance into level-4 task effectiveness

$O_{4,1}$: level-4 task sequences into level-1 interaction conditions

The purpose of level 7 is to specify the mission in terms of desired "ends" (i.e., the end state) along with a minimal specification of the "ways" (i.e., essential tasks and key tasks) of achieving the ends. Explicitly organizing the mission (including references to environment and location) into parts (essential and key tasks with associated purpose) and package (desired end state) sets the stage for a clear understanding of and linkage to the "means" (at levels 4 through 1) required to enable the ways and achieve the ends.

Level-7 parts consist of essential tasks (with associated purposes) and the commander's intent; they correspond essentially to what might appear in operational orders or plans (OPORDs/ OPLANs). The combination of level-7 "parts" that make up an overall mission is called a *mission-specification package*. Taken together with references to associated specification packages for level-6 environment and level-5 location and time, it represents the mission part of the MMF. As an example, a mission statement might be "deploy to theater and conduct forcible entry in order to establish a lodgment for follow-on forces to disembark, assemble, and prepare to conduct offensive operations against insurgents." The commander's intent might be "be prepared to conduct continuous operations without resupply for 48 to 72 hours. Prevent threat forces from disrupting debarkation and assembly of follow-on forces until transfer of responsibility occurs. The end state for this mission occurs with transfer of responsibility to follow-on forces for the lodgment area."

The means by which missions are defined and accomplished are collectively represented by levels 1–4 and the four operators. In specifying the means, we often begin at level 4 with decomposing the mission into necessary tasks to achieve the desired results. Depending on the objectives and questions for a particular analysis or study, decomposition can begin at any level of war from the strategic national level on down, and it can extend down to any level required. Decomposition breaks the mission down into small vignettes and related tasks that can be assigned to units, individual Soldiers, and platforms. Each task is then assessed to determine the required capabilities of personnel and functionality of materiel. For each such capability identified, conditions and standards are considered within either a generic or scenario-specific operational environment. To identify DOTMLPF availability or gaps, we enumerate and aggregate for comparison the capabilities and functions (supported by the relevant doctrine, organization, training, materiel, leader development, and facilities solutions) required by the mission.

1.3 Demonstration of the MMF Implementation

The main thrusts of this demonstration were to show that the pieces of the MMF could be modeled to:

- describe a mission by tasks, conditions, and standards;
- describe platform elements in high resolution by components and fault trees such that when interactions occur, the specific state change can be explicitly calculated to re-evaluate current performance and capabilities; and
- model adaptive decision making to select alternative courses of action (COAs) and to reallocate resources.

The demonstration consisted of running a vignette through a scripted discrete-event model known as the *Storyboard Model* (SBM). The model had a small amount of randomness introduced by using stochastic methods to determine, for example, the time required to acquire targets, send messages, and respond to calls for fire, and the delivery errors of weapons. Components' state was changed with a simple script specifying the timing for component failures, damage, and repair. The script used exponentially distributed random numbers with specified rates of failure and repair. To vary the outcome of the stochastic effects, we ran multiple copies of the state-change script through the model. The required tasks in the vignette were also scripted, but deterministically.

Much of the input data was fictitious or at best surrogated, and the scripted nature of the vignette strongly constrained the actions in the simulation. Neither limitation is significant,[*] though, because the study was not intended to address particular analytical issues themselves but rather to demonstrate the feasibility of structuring analyses according to the MMF.

[*]The limitations were due mainly to inavailability and classification of data and to the short time available to do the study.

The initial idea for a model was to build a simple one-sided constructive simulation of a short fire-support vignette. We chose fire support for two reasons. First, ARL expected that the Non-Line-of-Sight Cannon (NLOS-C) would be one of the first Future Combat Systems (FCS) platforms for which detailed platform-level data would be available. Second, fire support involves several platforms: some for surveillance/target acquisition, others for fire mission planning and decision making, and yet another or others to execute the fire mission. Such a vignette would support exploring the utility of the MMF to SoS analysis better than a purely direct-fire vignette, where the mission might be accomplishable without significant cooperation across platforms. In the end, however, the operational subject matter experts (SMEs) who developed the vignette did include some direct-fire elements as peripheral players that neither engaged targets nor worked any tasks besides their own movement.

The vignette was fairly one-sided, and this led to an emphasis throughout the demonstration on the friendly force, or "own" force (OWNFOR). By focusing on the execution of typical tasks for a mission, the model was able to track the impact of platform capability changes over time, and hence the entire small-scale SoS. Specifically, it tracked those capability changes that were caused by combat damage, component failure, repair, and resupply. The opposing force (OPFOR) was included merely to provide a load on the OWNFOR capability for surveillance and target acquisition and to serve as targets for the OWNFOR fire-support assets. Acquiring OPFOR targets also added a load on the OWNFOR communications network above and beyond its routine situation awareness (SA) messages.

1.4 Relationship of the MMF to Mission-Based Test and Evaluation (MBT&E)

Army evaluations have traditionally been based largely on technical performance requirements that may not correlate well to current wartime missions and operational context for testing. DOT&E and ATEC want to ensure that the materiel provided to our Soldiers is suitable, sustainable, survivable, and effective in the missions expected during its service life. For that reason, ATEC has developed the policy and processes of MBT&E—essentially an application of the MMF to testing and evaluation—which aims to develop a methodology for assessing the mission impact of systems' failure to meet technical performance requirements or being degraded. Seen in the light of the MMF, MBT&E methodology aligns system components and functions to a specified tactical mission at a higher resolution than the traditional M/F LoF [*8*]. Use of the MBT&E processes enables us to evaluate systems in terms of not only technical performance parameters, but also the capability requirements imposed on the systems by operational missions.

2. Demonstrating SoS Analysis With the MMF

The fundamental processes for demonstrating the use of the MMF for SoS analysis fall into three areas: defining the mission context, developing the model and inputs, and analyzing the model results.

2.1 Context Definition

The operational context for modeling was developed by a group of military experts from the Dynamics Research Corporation (DRC) via mission decomposition. As discussed in section 1.1, mission decomposition is the key process for demonstrating the MMF levels 7–4, and we used several phases of that process: analyze the research and mission, develop the vignette and mission threads, identify appropriate mission tasks and the system capabilities that they require, and apply the military decision-making process to develop COA.

For the mission analysis and research, we focused on the FCS systems and their components, in particular, the mounted combat system (MCS) company, support elements from the unit of action (UA) and combined arms battalion, and the NLOS-C battalion; the details are provide in appendix B. We also considered the characteristics and components of mounted and dismounted threat infantry forces [*11–13*]. In developing the vignette, we began by selecting one scenario from a set produced by the Unit of Action Maneuver Battle Lab and from it built the road to war* and created the tactical environment for the demonstration. We scaled down the scenario, using only MCS Company A (MCS A) supported by an NLOS-C battery. We conducted a step-by-step analysis of the operation, phase by phase, using the action-reaction-counteraction process to sequence the tasks to be performed. This analysis also determined the purpose and helped define the relevant measures of effectiveness and measures of performance for each task to be performed by an FCS platform. We then developed an execution matrix to depict the actions of MCS A over the 8-hour period by the phase and battlefield operating system as well as the outcome of the war game. The details of the scenario and the execution matrix are provided in appendix C.

To identify mission tasks and develop the mission threads, we used the scenario generation common operating environment tool of the Joint Training Information Management System. Task attributes included duration, triggers for start and stop, dependencies, and interrelationships. To make the vignette into a script that could be modeled as a sequence of tasks, we developed a time-ordered events list (TOEL) enumerating what would (or might) happen in the vignette. The TOEL included 31 distinct tasks. We then further refined the events

*The *road to war* is a fictitious but plausible history of the events that led to the initial conditions in a scenario or vignette. It is a standard part of TRADOC scenarios and gives background to the motives and objectives of the participants.

on this list into lower-level tasks that each platform (or small unit) would execute as a function of time or situation to accomplish the mission.

We next converted the TOEL to Gantt charts so that we could analyze a cross section of platforms performing tasks at any given time. The Gantt charts were also helpful to script the enemy interactions that would degrade the platforms' capabilities.

Having identified the appropriate tasks with conditions and standards, we derived the required system capabilities by engineering analysis and emulating the MDMP. Following the MMF, we then developed task-based fault trees to translate (1) the effect that changes in component state have on the task(s) being performed by the platform/system, (2) the effect that failures in platform-level tasks have on supported collective tasks, and (3) the effect that failures at essential collective tasks have on the mission.

2.2 Model Development

The SBM was a rapid-prototype development written in C++.[*] Because the vignette was largely one-sided, the modeling emphasized the OWNFOR to demonstrate how changes in low-level state could be tracked over time and to show how changes in the capabilities of each OWNFOR platform (and hence of the entire small-scale SoS) affected accomplishment of mission tasks.[†]

We also developed a number of purpose-built pre- and postprocessors (in C, C++, AWK, and Java): generator of component status vectors, $O_{2,3}$ mapper, statistical post-processor, graphical display, and vignette engine (the core executable code of the SBM). An overview of the model's process flow is shown in figure 1.

[*]We developed approximately 10,000 lines of new code. This new code also leveraged software previously developed by Richard Saucier [*14*] (Random class) and Richard Sandmeyer (bit manipulation and event scheduling software).

[†]Capability changes caused by combat damage, component failure, repair, and resupply affected the extent to which those tasks could be accomplished.

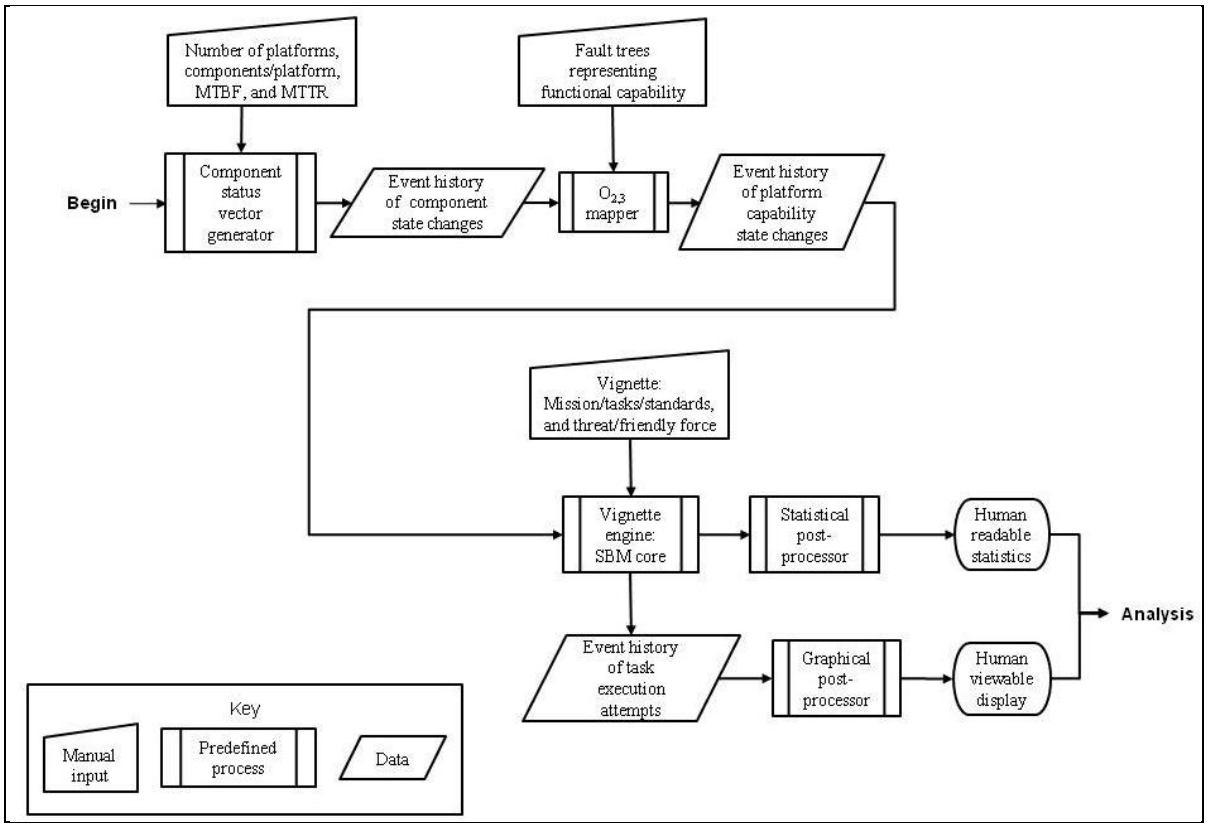


Figure 1. Storyboard modeling process.

2.2.1 MMF Levels 1–3

For expediency, changes in the state of platform components were not generated dynamically, but simply scripted from an external event file. This event file was produced with two of our preprocessors: one generated the changes in level-2 component state and the other was the $O_{2,3}$ mapper utility from MUVES, the Army's standard ballistic vulnerability/lethality model.

More specifically, at level 2, a simple event-sequenced model was written to generate for each platform a script, or history of the component* state changes and personnel incapacitation, over a specified number of replications of the vignette. User-specified inputs are expected time between combat damage events and the mean number of components killed per such event, mean time between component failures (MTBF), and mean time to repair (MTTR) each component. The event-sequenced model requires other information such as the number of components for each platform, the number of platforms of each type, random number seeds, and the number of vignette replications to be generated. This preprocessor then generates a (time-ordered) sequence of random events (using an exponential distribution of inter-event times for each event type).

*For this purpose, a component means a critical component. That is, a component that is relevant to at least one system-functional capability. A component is irrelevant to a capability if its state (whether functional or dysfunctional) never affects whether the capability fault tree is cut.

To determine the impact of component damage on platform functionality, we identified relationships between component and function and modeled them using the MUVES $O_{2,3}$-mapper program.* This utility program uses the sequence of component state vectors to evaluate the fault tree representation of platform-functional capability. The output generated is a time-ordered sequence of capability vectors in which each entry includes a time and a potential change in the platform's functionalities that occur at that time.

2.2.2 MMF Levels 3 and 4

Army operational SMEs reviewed at level 4 the 31 tasks in the vignette and then the level-3 functional capabilities of each platform. The review identified the system-specific functions whose loss would cause the platform to fail at some task. We represented the correspondence between task requirements and platform capabilities not by an input file to SBM, but by hardwiring (object-oriented) methods.†

2.2.3 MMF Levels 5–7

The core of the SBM is the vignette engine, which models the MMF levels 5–7. It is a scripted discrete-event model. Perhaps a good way to describe the model is to first describe the entities or objects that it models and the events that potentially change their states. Conceptually, the model entities can be grouped into two classes: OWNFOR (or Blue) platforms and OPFOR (or Red) units.

Each OWNFOR platform—hereafter called simply a *platform*—is a composite of up to four object types: a mover, a sensor suite, a communications node, and a shooter. (Not all platforms have all four object types; for example, the unmanned aerial vehicles (UAVs) used in this vignette had no shooting capability.) Each platform maintains its own perception of the situation based on the common operating picture. Each has its own physical state based on the components that have failed, been damaged, or been repaired at each instant in time and also has a corresponding set of *elements of capability degradation* (ECDs) based on the component states. The details of the platform representation are provided in appendix E.

2.3 Model Analysis

In analyzing the model's behavior we had two objectives: to verify that the model and postprocessors executed as expected and to determine what information was useful for SoSA.

*The mapper program is an AWK-language script developed to read the component state changes from their generator and make a system call to the $O_{2,3}$ mapper of MUVES. The script then formats and outputs the results for each call to be used by the SBM.

†The fault trees were translated into Boolean expressions in the C++ code for the functions. Then the true-false value corresponding to each function in the current platform state was substituted into the expression via function arguments. The value of the expression then indicated whether the fault tree for the task of interest was cut or intact.

2.3.1 Did the Codes Execute Correctly?

One of the SBM products is a log file to monitor and determine whether the model appears to be running correctly. Specifically, does the behavior of the players in the simulation seem reasonable and is the logic played as expected or were there programming flaws requiring correction?

After verifying that the SBM model ran correctly, we ran the simulated vignette for multiple replications (typically 10) to provide data to feed our examination of the postprocessors.

The SBM produces an event-history file used by a map-display program implemented as a postprocessor (figure 2). This post-processing begins with a display of the initial positions of the OWNFOR and OPFOR. As it then plays the vignette back, the program moves the platforms and units around the map. It also shows sensors' fields of view, message traffic, and weapon firing. The postprocessor display is a simple graphical view of the vignette that also shows the effect of degradations. For example, one can watch immobilized platforms come to a stop, degraded sensors having fields of view reduced (or eliminated), degraded weapons that did not fire, degraded communicators that did not send messages, and catastrophically killed platforms ceasing to function altogether.

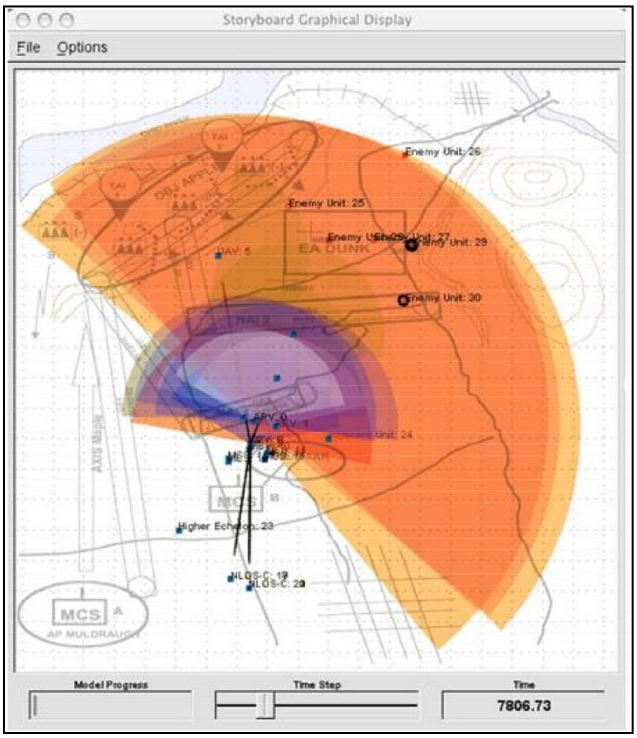


Figure 2. Playback graphical display.

Another graphical postprocessor allows one to watch the health of the platforms and the entire OWNFOR change as the vignette is executed. This display (shown in figure 3), also implemented as a postprocessor, is driven by a health-history file. The history file provides an

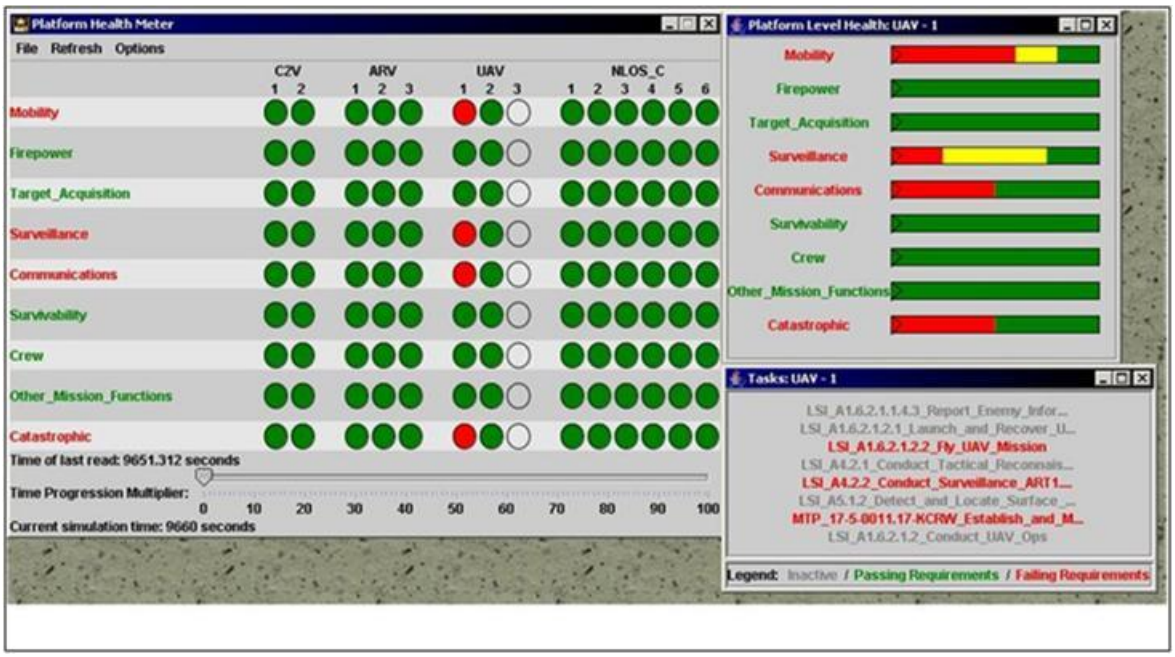


Figure 3. Health meter display.

instantaneous comparison of each OWNFOR platform's capability against the requirements of the tasks currently demanded of it.* There is also an aggregate (or collective) OWNFOR health assessment showing the instantaneous capability of the force to perform the tasks required of its mission.

The health meter/health bar display can be executed in synch with the previously described map display. This allows one to watch the platforms move about the map and observe the changes in their health. One could pick a point in time to examine several aspects of the vignette at a platform level, i.e., what is the extent of capability (full, partial, or zero), the state of repair, and the platform's ability to execute the mission? For example, if a platform came to a stop, one can look at the health information for that platform to see whether it lost mobility or was still mobile but just reached a point in the vignette where it was scripted to stop. The details of determining the health of collective tasks are presented in section H.3 of appendix H.

2.3.2 What Information Is Useful for SoSA?

The playback utilities can only be used to examine a single iteration of the simulation at a time. To analyze multiple iterations, statistical analysis proved more helpful than the playback utilities for evaluating SoS collective task allocation, task and mission execution, and the impact of

*_Instantaneous_ means that the display shows whether the capabilities at each instant were sufficient to perform the tasks required of the platform at that instant. This is not the same as showing whether the platform successfully completed the task. The platform may have been required by the vignette script to be capable of maintaining surveillance of an area for an hour, but even if it were incapable for some of that time, it may still have been capable long enough to acquire and call for fire against the OPFOR units in that area. Therefore, instantaneous capability does not correlate perfectly with task success or failure.

system capabilities on mission completion. By using the formality of collective tasks, one can assess the efficacy of the SoS.

To determine the effect that state changes have on the mission, we collected varying levels of information during model execution. The model collects eight types of data at regular intervals for statistical calculations and reporting at the end of each run.* Five of the data types consisted of correlation tables and related information. The other three data types were the ECDs' relative frequency of occurrence by platform type, tasks' pass/fail rate, and causes of mission failure.

The correlations and related information are listed as follows:

1. Mission success/failure correlated with task pass/fail for every task type.
2. Mission success/failure correlated with ECD for each ECD and every platform type.
3. Mission success/failure correlated with component state (functional/dysfunctional) for every component type on every platform type.
4. Task pass/fail correlated with ECD for each ECD and every platform type.
5. Task pass/fail correlated with component state for every component on every platform type.

The correlation tables and related information are presented in similar formats for all five data types. First, there is a label indicating the level of information for comparison: mission, specific task, ECD, or component. The label is followed by a line giving the conditional probability $P(F_1|X)$ of failure F_1, where X is any of failure of another task F_2, presence of an ECD, and dysfunction of a component. Table 1 presents an example correlation table showing the impact of system capability on mission completion. In this case, the data are for the ECD *M3* on the command and control vehicle (C2V) platform type (which included both the C2V and its backup). The Monte Carlo estimate of mission failure occurred with a relative frequency of 0.874 when a platform of type C2V was suffering ECD *M3*.

*The model calculates statistics representing the entire set of vignette replications for a run. To obtain statistics for a single replication of the vignette, it is simple to run just that one replication.

Table 1. System capability impact on mission completion.

Mission versus ECD: state 2 (m3) on platform type 3 (c2v)		
P (mission failing /task failing) = 0.874172		
Sample size = 9600		
Raw data		
132	19	
7562	1887	
	Mean	Standard deviation
X =	0.9842710	0.1244260
Y =	0.1985420	0.3989020
Covariance of X and Y =	0.0011437	
Correlation of X and Y =	0.0230434	

Next is a two-by-two table of the raw sample data. If one considers a hierarchy from high to low consisting of mission, task, ECD, and component, then the two columns correspond to the states in the higher level in the hierarchy, and the two rows to the two states in the lower level. In the example shown in table 1, the columns are mission success and the columns are capability retention, so of the 9600 cases (over 10 replications), there are 132 cases where the mission is failing and the ECD is in effect, 19 where the mission is succeeding and the ECD is in effect, 7562 where the mission is failing but the ECD is not in effect, and 1887 where the mission is succeeding and the ECD is not in effect.*

The sample mean and standard deviation for *Y* (higher level in the hierarchy, mission, or task) and for *X* (lower level, task, ECD, or component) are then printed out, followed by the sample covariance and the sample correlation.

When we treat the conditions as Boolean random variables, the sample statistics results have a mean of 0.984 and standard deviation of 0.124 for *M3* not occurring, and a mean of 0.198 and standard deviation of 0.399 for mission success. The covariance and correlation values of the two random variables are so low that one should not expect this ECD to account for many of the mission failures. This was not surprising given how rarely this ECD is in effect.

After reporting the correlations, SBM prints table 2, which contains the mean fractional time during which each platform experienced the corresponding degradation of capability.

*There was a change of convention here. Since an ECD's occurring means the loss of a capability, "true" could have meant either that the ECD had occurred or that the capability was still operational (i.e., it had not suffered the corresponding degradation). The latter convention was used in this demonstration.

Table 2. Time spent in a degraded capability state.

	M1	*M2*	*M3*	*M4*	*F1*	*F2*	*F3*	*F4*	*F5*	*A1*	*A2*
ARV	0.04	0.08	0.01	0.06	—	—	—	—	0.02	0.00	0.00
	0.12	0.15	0.04	0.10	—	—	—	—	0.05	0.04	0.03
UAV	0.11	0.23	0.11	0.25	—	—	—	—	—	—	—
	0.25	0.27	0.25	0.25	—	—	—	—	—	—	—
C2V	0.04	0.02	0.02	0.12	0.00	—	—	—	—	—	—
	0.13	0.12	0.02	0.12	0.00	—	—	—	—	—	—
NLOS-C	0.06	0.10	0.03	0.08	—	0.01	0.10	0.09	0.04	—	—
	0.12	0.16	0.04	0.08	—	0.06	0.12	0.12	0.06	—	—

In the table, each double row corresponds to an OWNFOR platform type and each column corresponds to an ECD. The upper number in each row-column intersection is the fraction of the vignette time that a platform of the given type experienced the given ECD. For example, over the 10 replications of the vignette in this run, the two identical platforms of type command and control vehicle (C2V) (hereafter referred to as *C2V-1* and its backup *C2V-2*) spent an average of 2% of their time suffering *M2* (reduced maneuverability). The lower number, 0.12, indicates that when implied states were included, the time spent suffering *M2* was 12%.*

This table allows the analyst to see which ECDs are likely to cause frequent force degradation (an example shown in appendix H-9 is repeated here for convenience). As with any of these statistical measures, the data is usually only an indicator of possible influence and not a guarantee of a causal relationship.

Table 3 shows an excerpt from the task pass/fail rate collected over the replications of the vignette in a run (10 replications in this case). There are 31 different tasks in the vignette with authoritative number assignments and descriptors: LSI, AUTL, or UJTL. For each task, a count is tallied for the number of times the task is sampled and the platform has sufficient capability to perform the task (a "pass"), followed by the number of times the task is sampled and the platform does not have sufficient capability to perform it (a "fail"). These counts are then converted to fractions.

* An ECD can occur either because some dysfunctional component causes a fault tree to be cut or because the ECD was implied by the occurrence of another ECD. In the example, *M2* was implied whenever *M4* occurred.

Table 3. Task execution and failing rates.

Task No.	Task ID, description, and assigned platform	Passing		Failing	
		Absolute	%	Absolute	%
0	ART 3.3.1.1, conduct surface-to-surface attack by NLOS_C	1070	89	130	11
1	ART 7.2.5, disseminate common operational picture and execution information by C2V	995	99	5	1
2	LSI A1.2, conduct tactical maneuver ART 2.2 by C2V	1560	78	440	22
3	LSI A1.2, conduct tactical maneuver ART 2.2 by NLOS_C	7323	97	237	3
4	LSI A1.6.2.1.1.4.3, report enemy information by ARV	95	95	5	5
5	LSI A1.6.2.1.1.4.3, report enemy information by ARV	900	100	0	0

These data are reported by platform type, not by individual platform. Also, the samples are taken only for times when the task is required of a platform. For example, task number 8 is sampled a total of 7640 times (5190 + 2450) during the vignette. Since this particular run consists of 10 vignette replications, each replication contributes 764 samples. Since this task is performed by the UAV type of platform of which there are three in the vignette, each UAV is sampled an average of 254.67 times during the run of the vignette to determine whether it has the capability to perform this task. This task fails almost one-third of the time in the vignette (0.3207 to be more precise) because on many replications, a catastrophic failure or damage takes out one of the UAVs.* It is also important to understand that these are time-weighted statistics in that a task may be required of a platform for 60 min at one time in the vignette and for only 5 min at a later time. In that case, the former requirement is sampled 60 times, and the latter only 5 times; consequently, as far as this statistical summary is concerned, having capability throughout the earlier (and longer) period appears more important than having it during the later period.

The output shows clearly that some tasks are sampled rather infrequently, whereas others are almost constantly required of their platform type. Thus the table reflects not only the time-weighted pass/fail capabilities of the platforms, but also how long during the vignette they are required. Of course, how long a task is required does not necessarily indicate its importance to mission success. A routine task may be required for the entire duration of the vignette, and an important message may require communications capability for only a tiny fraction of the vignette; but each may be essential to the ultimate success of the mission.

If the model had used real data instead of fictitious data, this table would allow the analyst to see which tasks were failed for the most time. That information could guide doctrine developers in modifying the mission's task composition to reduce the mission's dependence on the task and guide platform designers in improving the materiel to increase the capabilities' availability for the task.

*For a UAV, a total immobilization means that it crashed, not that it stopped and is awaiting repair like a ground platform.

The next SBM output (as illustrated in table 4) shows for each task the amount of time during which some capability that it would demand is unavailable.* Using the same task identification numbers as in table 3, it presents the total number of failures (samples where the platform is found to have inadequate capability to perform the task). For example, task 7 is unachievable and mission critical on 2185 samples over the 10 replications of the vignette. If one or more platforms of the relevant type are incapable of performing the task at a time when it is considered critical, it counts as a failure for this tally; if all platforms of the relevant type are capable of performing the task or if the task is not considered critical at the sampled time, then it is not counted as a mission failure. During 0.4552 of the vignette duration, there is at least one UAV incapable of performing task 7 when critical. This fraction is taken over the entire vignette duration, and at many times during the vignette multiple critical tasks are unachievable.†

Table 4. Examining mission failure without task reallocation.

Task No.	Number of Samples	Percent of Vignette Duration
0	40	0.001
1	5	0.002
2	405	0.084
3	51	0.010
4	5	0.001
5	35	0.007
6	52	0.010
7	2185	0.455
8	286	0.059
9	271	0.056
10	1107	0.230
11	1774	0.369
12	15	0.003
13	57	0.011

The vignette variation modeled for table 4 does not take into account redundancy of capability across platforms—one of the benefits of the SoS. When one platform is failing a mission-critical task but there is another available platform capable of performing the task, tasks can be reallocated. Such adjustment to resources greatly reduces the frequency with which mission-critical tasks fail. Table 5 shows the results when redundancy and reallocation are taken into account.

*For example, the task might have been scheduled for an entire 30-min interval but completed in the first 5 min. In that case, a failure during the remaining 25 min would be scored as a significant failure time even though the task was successful and the mission unaffected by that failure.

†In this example, adding up the fractions of time would exceed 1.00.

Table 5. Examining mission failure with task reallocation.

Task No.	Number of Samples	Percent of Vignette Duration
0	12	0.002
2	35	0.007
3	2	0.000
19	35	0.007
20	35	0.007
22	35	0.007
24	66	0.014
25	259	0.054
28	10	0.002

The count is calculated over all replications of the vignette and incremented only when the sampling encounters a time when the task is critical, the platform has inadequate capability to perform the task, and there are not enough alternative platforms capable to take over the task. With resource reallocation the percentage of mission failure drops. Now task 7 does not show up as unachievable because whenever it's critical there are sufficient alternative resources to perform the task.

Statistics on mission failure are shown in table 6. The first line reports that on 3847 samples over the 10 replications, one or more platforms lack the capabilities required to execute one or more critical tasks, and that those samples constitute 0.80 of the vignette duration. When reallocation of resources is taken into account, the corresponding numbers drop to 376 and 0.08. The great improvement when resource reallocation is allowed shows the clear benefit of avoiding single points of failure in the mission planning.

Table 6. Mission failure.

Without adjustment, mission failing on	3847 checks	0.80 of time
With adjustment, mission failing on	376 checks	0.08 of time
Commander's intent achieved on 8 of 10 replications		

The commander's intent is achieved only if (1) at least one of the C2V and its backup, (2) at least seven of the MCSs, and (3) at least four of the NLOS-Cs are functional at the end of the vignette, and only if the MCS and C2V (or its backup) have reached their objectives. With reallocation, the commander's intent is achieved on 80% (8 out of 10) of the replications.

3. Follow-on Efforts

Although the demonstration involved only one simple vignette, it applied the MMF in a new way that includes the area of automated war-gaming. Since then, several follow-on efforts have successfully shown the utility of this approach.

In 2005, DRC applied the MMF approach to an Army map exercise (MAPEX) that examined a UA in stability and reconstruction operations (S&RO). The purpose of the exercise was to study such operations, identify the inherent tasks, and determine capability gaps or external support requirements that a UA would confront while executing S&RO—specifically, to determine capability gaps associated with an FCS-equipped Brigade Combat Team (BCT) conducting an S&RO mission with only its organic authorized equipment. The MAPEX was designed to exercise, in the context of a relevant TRADOC-developed operational scenario, the S&RO mission tasks documented in the operational mode summary - mission profile (OMS-MP) produced by the Armor Center/Unit of Action Maneuver Battle Lab (UAMBL). For each of the OMS-MP mission tasks, DRC developed a series of detailed task-based mission threads using as their authoritative source for the tasks the AUTL and the FCS single integrated task list. They applied current doctrine and tactics, techniques, and procedures (TTPs) along with emerging lessons learned from theater to identify subordinate and supporting tasks and then construct mission threads with appropriate task durations, interdependencies, and sequencing. In their final step before the MAPEX, DRC and TRADOC verified and validated the resulting mission threads with uniformed SMEs at UAMBL.

The MAPEX itself was a walk through the execution of the mission threads, with the DRC team facilitating. Discussion of the tasks with the SMEs whom UAMBL had assembled from each of the warfighting functions yielded information about the categories and levels of functionality required to enable each task. Because the UAMBL participants were also SMEs on FCS, the DRC team asked them to assess whether the manpower and equipment organic to the FCS BCT would suffice to accomplish each task under the scenario conditions. The facilitators recorded the discussion results in the notes tab for each mission thread task, which allowed the assembled SMEs to both correct in real time any perceived errors before they moved on and highlight any tasks with capability gaps.

UAMBL used the resulting annotated mission threads as the basis for identifying and documenting which desired capabilities required solutions from one or more of the DOTMLPF categories.

In 2006, supporting the joint warfighter test and training capability at the Aberdeen Test Center, ARL used the MMF to evaluate the explicit contributions of soldier performance and effectiveness from both an individual as well as a collective perspective in the context of a

Future Brigade Combat Team tactical operations center (TOC). This study resulted in the development of an executable model (using a discrete event simulation environment called EXTEND) to analyze how the accomplishment of individual, system, and collective tasks by a complex SoS (the TOC) was affected by incapacitation of humans and degradation of technological components.

In most cases where the MMF is applied to support analysis, whether for T&E, experimentation, science and technology tradeoff, or analysis of alternatives, the operational context already exists in the form of doctrinally developed plans and orders with associated graphics. For executing and assessing operations, these doctrinal products are both necessary and sufficient for well-trained military planners and operators because they have years of training and experience with the terms and formatting used and they implicitly understand the linkage between certain terms and the underlying TTPs associated with them. But these products are not sufficient for use by scientists, engineers, analysts, software developers, etc., who—for want of detailed understanding of the mission/operational context—cannot by themselves properly assess potential impact on mission/operational effectiveness. A major benefit of the MMF is that with it, those who lack the operational expertise can rely on trained and experienced SMEs who can analyze existing doctrinal scenario products (e.g., OPORDS/OPLANS); the results they generate can then be used in analysis, modeling, and simulation, and also reused for follow-on study and analysis. Depending on requirements of the supported analytical objectives, the operational SMEs can generate such MMF products as:

- task-based mission threads with associated time-ordered event list
- mission-task lists with associated conditions and standards
- relational databases with tables organized in accordance with MMF levels and operators
- detailed process models
- discrete event simulation models with associated input files and results files
- computational models
- DOD architecture framework views (operational, systems, and services views)

4. Summary

This demonstration illustrates that the MMF offers a useful and effective framework for evaluating the effectiveness, suitability, and survivability of complex warfighting systems in the context of their contribution to operational missions. The MMF also avoids some known problematic approaches that the analysis community has used for decades. In addition, by using

the MMF with its MDMP formalism, we are able to clearly examine mission outcomes in warfighter parlance, including new metrics and mission-based effectiveness.

We demonstrated the collection of three kinds of data: (1) damage or loss history of personnel and component parts, (2) the concomitant degradation of platform-level capability, and (3) the inability to perform mission tasks. Also demonstrated was the analysis of these new metrics with three basic methods: log file review, playback of event and health history in a visual display, and review of statistical data.

The SBM vignette engine demonstrated a method of tracking the impact of component changes over time on system capabilities. Correlating platform capabilities to standard tasks allows the model to track task execution and the impact on mission effectiveness. The statistics computed by the model illuminate the correlations among success or failure of the mission, success or failure of tasks, state of platform capabilities, and state of platform components. They similarly illuminate the relative frequencies of degradations in capabilities, the rate of task success and failure, and the likely causes of mission failure.

5. References

1. Sheehan, J. H.; Deitz, P. H.; Bray, B. E.; Harris, B. A.; Wong, A. B. H.; The Military Missions and Means Framework. *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2003*, Orlando, FL, 1–4 December 2003.

2. Deitz, P. H.; Ozolins, A. Computer Simulations of the Abrams Live-Fire Field Testing, *Proceedings of the XXVII Annual Meeting of the Army Operations Research Symposium*, Ft. Lee, VA, 12–13 October 1988. Also BRL-MR-3755; U.S. Army Ballistic Research Laboratory: Aberdeen Proving Ground, MD, May 1989.

3. Deitz, P. H.; Starks, M. W.; Smith, J. H.; Ozolins, A. Current Simulation Methods in Military Systems Vulnerability Assessment. *Proceedings of the XXIX Annual Meeting of the Army Operations Research Symposium*, Ft. Lee, VA, 10–11 October 1990. Also BRL-MR-3880; U.S. Army Ballistic Research Laboratory: Aberdeen Proving Ground, MD, November 1990.

4. Deitz, P. H.; Starks, M. W. The Generation, Use, and Misuse of "PKs" in Vulnerability/ Lethality Analyses. *Proceedings of the 8th Annual TARDEC Symposium*, Monterey, CA, 25–27 March 1997. Also ARL-TR-1640; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, March 1998. Also *Journal of Military Operations Research* **1999**, *4* (1), 19–33.

5. Deitz, P. H.; Reed, H. L., Jr.; Klopcic, J. T.; Walbert, J. N. *Fundamentals of Ground Combat System Ballistic Vulnerability/Lethality*; American Institute of Aeronautics and Astronautics: Reston, VA, 2009; appendix E.

6. Headquarters, Department of the Army. The Military Decision-Making Process. In *Staff Organization and Operations*; FM 101-5; Washington, DC, 31 May 1997.

7. Ward, B. S.; Durda, D. *Combined Arms and Support Task Force Evaluation Model (CASTFOREM) Combat Simulation Via Capability States Vulnerability Methodology (CSVM)*; ARL-TR-2522; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, October 2001.

8. Apicella, F. J.; Wyan, K. W.; Wilcox, C. M. ATEC Initiatives in Response to the Office of the Secretary of Defense Policy Guidelines for Test and Evaluation. *International Test and Evaluation Association (ITEA) Journal* **2009**, *30*, 361–368.

9. Ward, B. S. Modeling and Simulation for Mission-Based Test and Evaluation (MBT&E). National Defense Industrial Association; 27th Annual National Test and Evaluation Conference, Tampa, FL, 14–17 March 2011.

10. Nelson, M. K. *Implementation of Mission-Based Test and Evaluation Methodology: Expended Efforts and Expected Benefits*; Department of Business, Organizations, and Society, Franklin and Marshall College: Lancaster, PA, October 2009.

11. Operational Requirements Document for the Future Combat Systems (ORD) UAMBL Fort Knox, KY, version 3, 14 April 2003.

12. Headquarters, Department of the Army. *O&O Plan Maneuver Unit of Action*; TRADOC Pam 525-3-90; Fort Monroe, VA, 30 June 2003.

13. Army Materiel Systems Analysis Activity (AMSAA). *Army Future Combat Systems Unit of Action Systems Book*, version 3.0, 22 May 2003.

14. Saucier, R. *Computer Generation of Statistical Distributions*; ARL-TR-2168; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, March 2000.

Appendix A. Missions and Means Framework

The U.S. Army Research Laboratory and other agencies within the U.S. Department of Defense* have formulated the Missions and Means Framework (MMF) as a structure to use in analyzing military operations.† The purpose of the MMF is to quantitatively evaluate alternative doctrine, organization, training, materiel, leader development, personnel and facilities (DOTMLPF) services and products.

Figure A-1, sometimes called the "left-right diagram," illustrates the important parts of the MMF approach (a one-sided view). The upper-left side shows the mission hierarchy and chain of command from strategic national (SN) command authority down to the tactical elements.‡ The tasks at each echelon become the mission(s) for its subordinate(s) until one finally reaches the level of individual platforms and Soldiers. These operationally relevant tasks are defined with associated conditions and standards. The platform and collective capabilities of the Soldier plus materiel shown on the lower-right side are defined. This side, within the hardware hierarchy, also shows the dependence of the Soldier, platform, and (ultimately) unit capabilities on the functional state of the materiel (subsystems and components) and human performance.

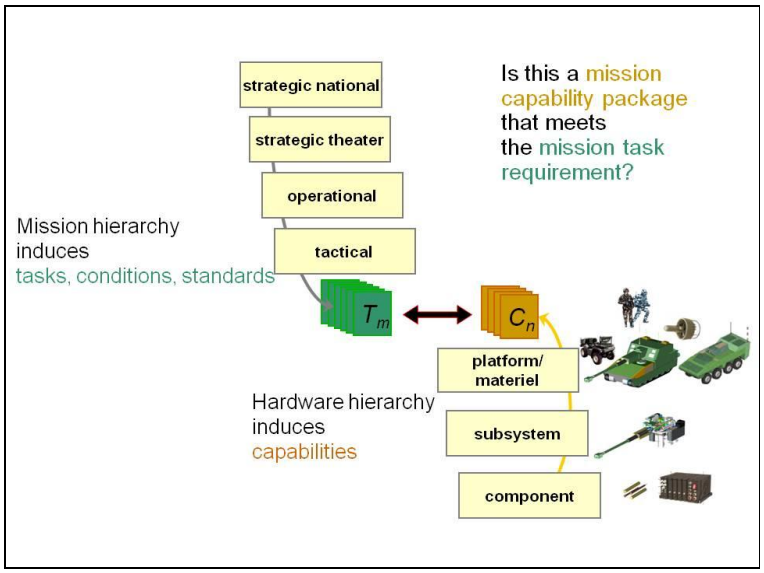


Figure A-1. The missions and means framework: what is it?

*Other MMF contributors include the U.S. Department of Defense Model and Simulation Office, the U.S. Army Materiel Systems Analysis Activity, and the U.S. Army Future Combat Systems (FCS) Combined Test Organization. In addition, the Dynamics Research Corporation, the Northrop Grumman Corporation, and the ORSA Corporation have been involved as support contractors.

†The operation might be a combat operation involving an opposing side, but MMF can also be applied to an operation in which no combat is expected to occur.

‡The actual number of echelons considered may differ from one analysis to another. If, for example, one side is a group of insurgents without a national command authority, there might be fewer echelons than in the figure. However, there will still be some mission(s) that can be decomposed into smaller tasks for the participating personnel and platforms on that side.

Mission decomposition is used to describe a mission in the most general terms down to the most specific. Decomposition can begin at any level of war to address the objectives and questions for the particular analysis or study, from the strategic national level or below, and extending to the lowest level required. Decomposition breaks the mission down into small vignettes and related tasks that can be assigned to units, individual Soldiers, and platforms. Figure A-2 is an example of mission decomposition from the MMF demonstration and can be broken down as follows:

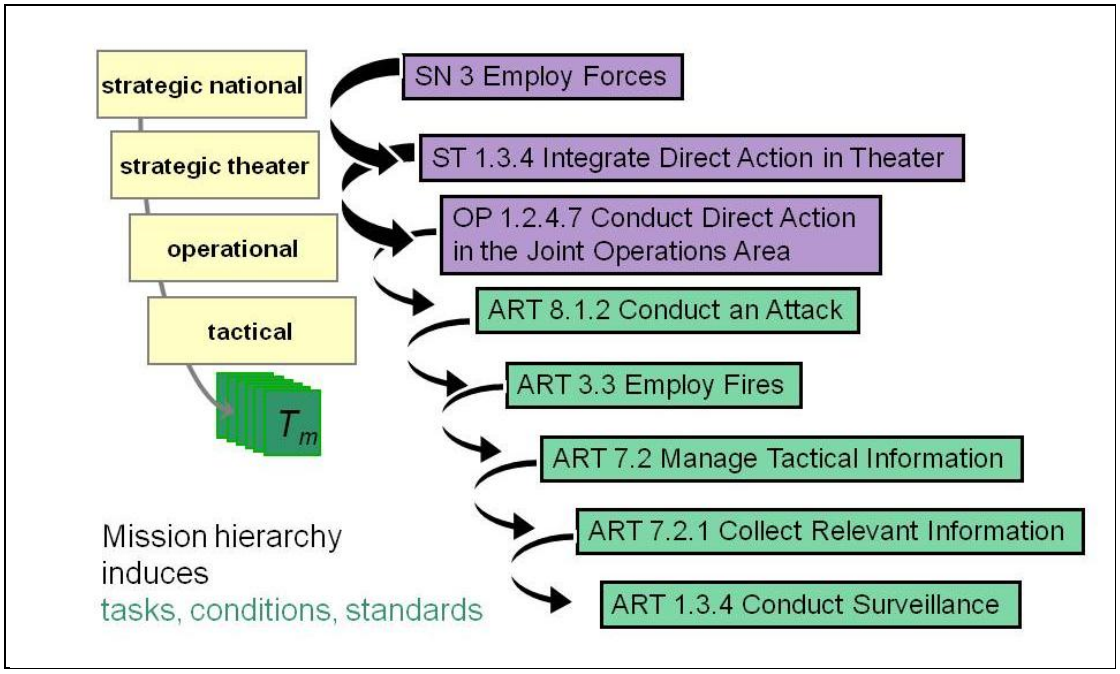


Figure A-2. Example of mission decomposition.

- Strategic National "SN 3 Employ Forces" represents the decision and action taken at the national level to use the military element of national power in response to a crisis caused by the actions of an external opposing faction.
- Strategic Theater "ST 1.3.4 Integrate Direct Action in Theater" represents the planning and coordination actions taken by the geographic combatant commander to capture elements of the opposing faction leadership based on emerging intelligence concerning their location and activities.
- Operational "OP 1.2.4.7 Conduct Direct Action in the Joint Operations Area (JOA)" represents the planning and execution actions being taken by the Joint Task Force Commander responsible for the JOA to effect the capture of the opposing faction leadership.

- Army Task "ART 8.1.2 Conduct an Attack" represents the mission given to an Army unit. In the MMF demonstration, the Army unit is an FCS-equipped company to support the Joint Task Force's action to capture opposing faction leadership. The purpose of the attack is to maneuver around opposing faction forces and occupy positions on key terrain to prevent those forces from interfering with the capture of opposing faction leadership by special operations forces.
- "ART 3.3 Employ Fires" represents the means by which the FCS-equipped company intends to engage opposing faction forces who might interfere with their maneuver to and occupation of the key terrain.
- "ART 7.2 Manage Tactical Information" represents the means by which the FCS-equipped company intends to maintain the situational awareness and understanding needed to maneuver out of contact with the enemy and employ fires to engage from standoff distances.
- "ART 7.2.1 Collect Relevant Information" represents the activity performed by the FCS-equipped company's Command and Control Vehicle to gather information from multiple sources on the current friendly and enemy situation. Much of the information gathered is expected to be transmitted digitally or by video feed from unmanned sensors and friendly force tracking systems.
- "ART 1.3.4 Conduct Surveillance" represents the activity performed by manned and unmanned sensors (i.e., UAVs) to monitor, detect, identify and report enemy activity in areas of interest.

Each task is then assessed to determine the required capabilities of personnel and materiel functionality. For each capability identified, conditions and standards are considered within either a generic or scenario-specific operational environment. To identify DOTMLPF availability or gaps, the capabilities and functions (supported by the relevant doctrine, organization, training, materiel, leader development, and facilities solutions) required by the mission are enumerated and aggregated for comparison.

The double-headed arrow at the center of figure A-1 is the heart of the MMF analysis; the comparison of the task(s) required to accomplish the mission against the capabilities available to perform them. Although this depiction is not new, there are several differences in the MMF from past analysis practices:

1. Standardized tasks: tasks at the platform and small-unit levels are defined by standard authoritative task lists rather than ad hoc for each analysis application.

   There were three lists of standard tasks used in the Storyboard Model for this demonstration: the Universal Joint Task List, the Army Universal Task List, and the Single Integrated Task List developed for FCS by the Lead System Integrator. Once a task was

clearly defined, operational experts identified the capability requirements of a particular small unit (or platform and personnel) to accomplish the task.

2. Higher-resolution data representation: platform functionality and personnel performance requirements are defined in much higher resolution than in traditional methodologies.

   Traditional methodology binned materiel platform states into five categories: undamaged, mobility-kill only, firepower-kill only, mobility-and-firepower kill (but still short of catastrophic kill), and catastrophic kill.*

   The MMF methodology defines platform and personnel states in terms of residual capabilities remaining after an interaction, such as combat damage, suffering casualties, reliability failures, or other non-ballistic interactions, such as electronic warfare. This higher resolution is especially important in analyzing system of systems (SoS). Multiple platforms having different capabilities (both initial and perhaps residual) may be required to execute a single low-level task. In cases where a set of tasks cannot be performed, the higher resolution of available capabilities can be used to better assess alternative courses of action (COAs).

3. Avoidance of data aggregation too early in analysis process: the decision as to whether a platform (or personnel) with given residual capabilities can accomplish a task is deferred until the task itself (with its requirements) is known.

   In previous methodology, a platform suffering some level of mobility loss is often considered "mobility killed" and thus unable to perform any task requiring mobility. When the MMF methodology is used, a platform with even a substantial mobility loss may still perform a task for which its remaining mobility is sufficient.

Figure A-1 can represent both the U.S. forces and its allies (own force [OWNFOR]) or opposing forces (OPFORs). Military operations change the state of materiel and personnel because of combat, wear, fatigue, illness, repair, and resupply. If an operation involves two (or more) opposing sides, then each side will seek to change the state of its opponent's materiel and personnel to hinder mission accomplishment.† The interaction of those forces is summarized in figure A-3.

*These are the categories used in many of the U.S. Army models and combat simulations. The underlying kill definitions used by the vulnerability analysis community are slightly different but are similarly limited in degrees of freedom. Another similarly limited categorization of wounded personnel states is used and would be superseded in MMF implementation.

†Opposing forces can influence each other via methods other than attrition to materiel and personnel (the right side of the figure A-2). Deception, for example, can affect the left side of the figure by influencing higher echelons to change the mission or the set of tasks deemed best to accomplish it.

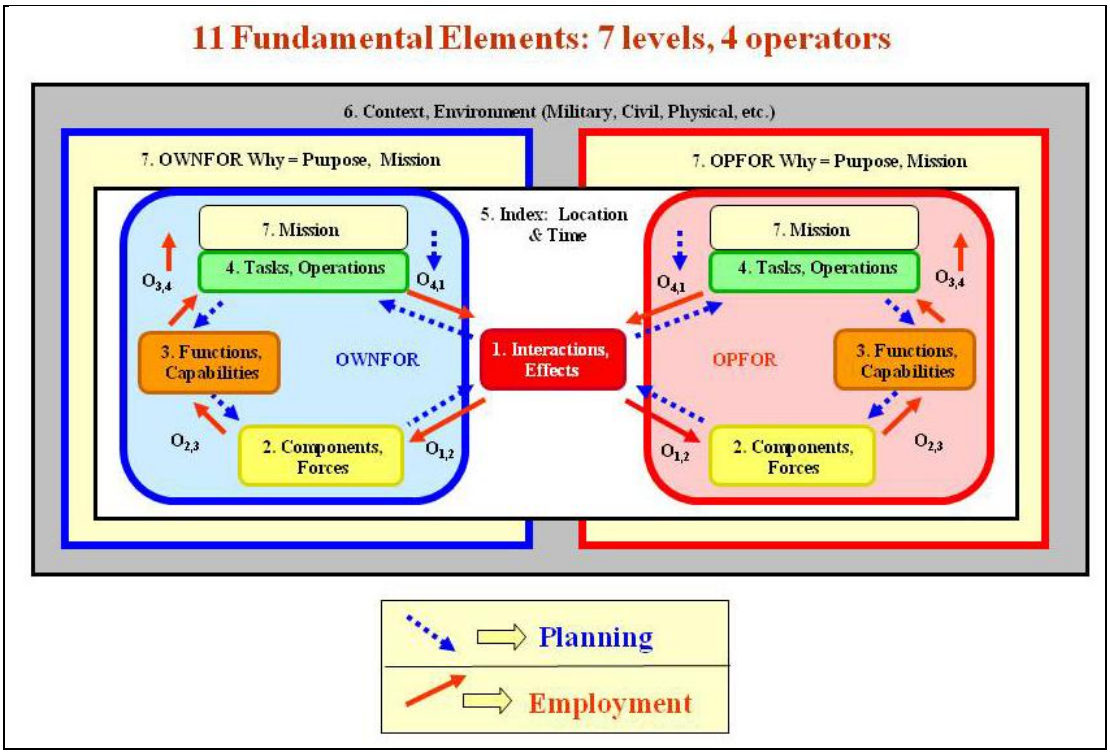


Figure A-3. A two-sided view of the MMF.

The two-sided view of the MMF uses 11 fundamental elements of content and transformations to organize and specify military operations. Content is organized into the following seven groups (hereafter called "levels"):

Level 7: Purpose and Mission

Level 6: Context and Environment

Level 5: Index and Location/Time

Level 4: Tasks and Operations

Level 3: Functions and Capabilities

Level 2: Components and Forces

Level 1: Interactions and Effects

In addition, the following four transformations (hereafter called "operators") are included:

$O_{1,2}x$: transforms level-1 interaction specifications into level-2 component states.

$O_{2,3}x$: transforms level-2 component states into level-3 functional performance.

$O_{3,4}x$: transforms level-3 functional performance into level-4 task effectiveness.

$O_{4,1}x$: transforms level-4 task sequences into level-1 interaction conditions.

The "Mission" part of the MMF is collectively represented by a level-7 Mission specification package, with references to associated level-6 Environment and level-5 Location/Time specification packages. The "Means" by which missions are accomplished are collectively represented by levels 1–4 and the four operators (hence the name Missions and Means Framework).

With levels 7–5 Mission specification package expressed as warfighting requirements within an operational context, the MMF derives levels 4–1 (with associated operators) the Means specification. The Means specification begins with mission decomposition into necessary tasks to achieve the desired results.

Each side is represented by four boxes in the diagram. The red box titled "1. Interactions, Effects" is common to both sides; others (those numbered 2, 3, and 4) are separate for each side.

The darker arrows show the direction of influence in planning.* Starting with box 1, each side's interactions affect the perception of the other side, causing it to consider or reconsider how to accomplish its mission (box 4, tasking and re-tasking). The tasking process itself takes into account the required functions to achieve the tasks (box 3), and lower-level planning decides how to accomplish those tasks with the resources (forces in box 2) available. Finally, those force components used to interact with the enemy make their own low-level plans to execute the interactions.

The lighter arrows going around the diagram in the opposite direction show the employment or physical effects. The interactions (box 1) cause state changes to both the OWNFOR and the OPFOR (damage to components on platforms, expenditure of resources such as fuel and ammunition, etc.) to affect the state of the forces (box 2). The state of the forces, in turn, directly determines the capabilities (box 3) of the force (and its constituent platforms and personnel) to accomplish various tasks (box 4). The tasks themselves then include how the OPFOR will be engaged (box 1 again).

During the unfolding of a scenario, each of the two sides causes changes in the state of the other's forces. This process affects the capability of each side to perform its mission-essential tasks and therefore may require a side to replan how it will accomplish its mission with its remaining resources.

The additional box numbered 5 represents the influence of time and place on the scenario; the box numbered 6 represents the environmental and political constraints that affect the scenario. Finally, the boxes numbered 7 (one for each side) represent the purpose and mission (goals) of each side at a high level. A second, smaller box 7 (Mission) is adjoined to box 4 on each side to emphasize the mission-to-task decomposition process as first shown on the left side of figure A-1.

*Perhaps "decision making" would be a better term than "planning" at some echelons since the choices made there among the range of possible responses may be almost reactive rather than carefully planned.

By breaking down the mission into simpler tasks in the MMF, SoS analysis benefits in several ways.

- Modeling: Allows for cleaner and simpler modeling alternative means of accomplishing tasks and COAs. One can model the resources required by the current task, quantify the current composite unit capabilities available to perform the task, identify the additional capabilities that the unit may need to request via the network, and study the changes in capabilities over time as platform and personnel states change.
- Analysis of SoS effectiveness: Track task execution to provide a completion rate for each task type over some range of scenarios, connect the cause of task failure (and ultimately mission failure when it happens) to the capabilities that were inadequate, and focus future analysis and experimentation on areas requiring corrective action.

The MMF fits well within the established policies and procedures of the Joint Capabilities Integration and Development System (JCIDS) in identifying, assessing, and prioritizing joint military capability needs. The comparison of the required capabilities determined during task analysis to the available capabilities determined during capability assessments identified gaps (analysis and assessment portion of JCIDS) that require DOTMLPF solutions to fill (reconciliation and recommendation portions of JCIDS).

INTENTIONALLY LEFT BLANK.

Appendix B. Unit Organization

The Army unit represented in the model was a Future Combat System Mounted Combat System (MCS) Company with support elements from the Unit of Action and Combined Arms Battalion, and the Non-Line-of-Sight Cannon Battalion. Figure B-1 displays the unit organizational chart for the MCS.

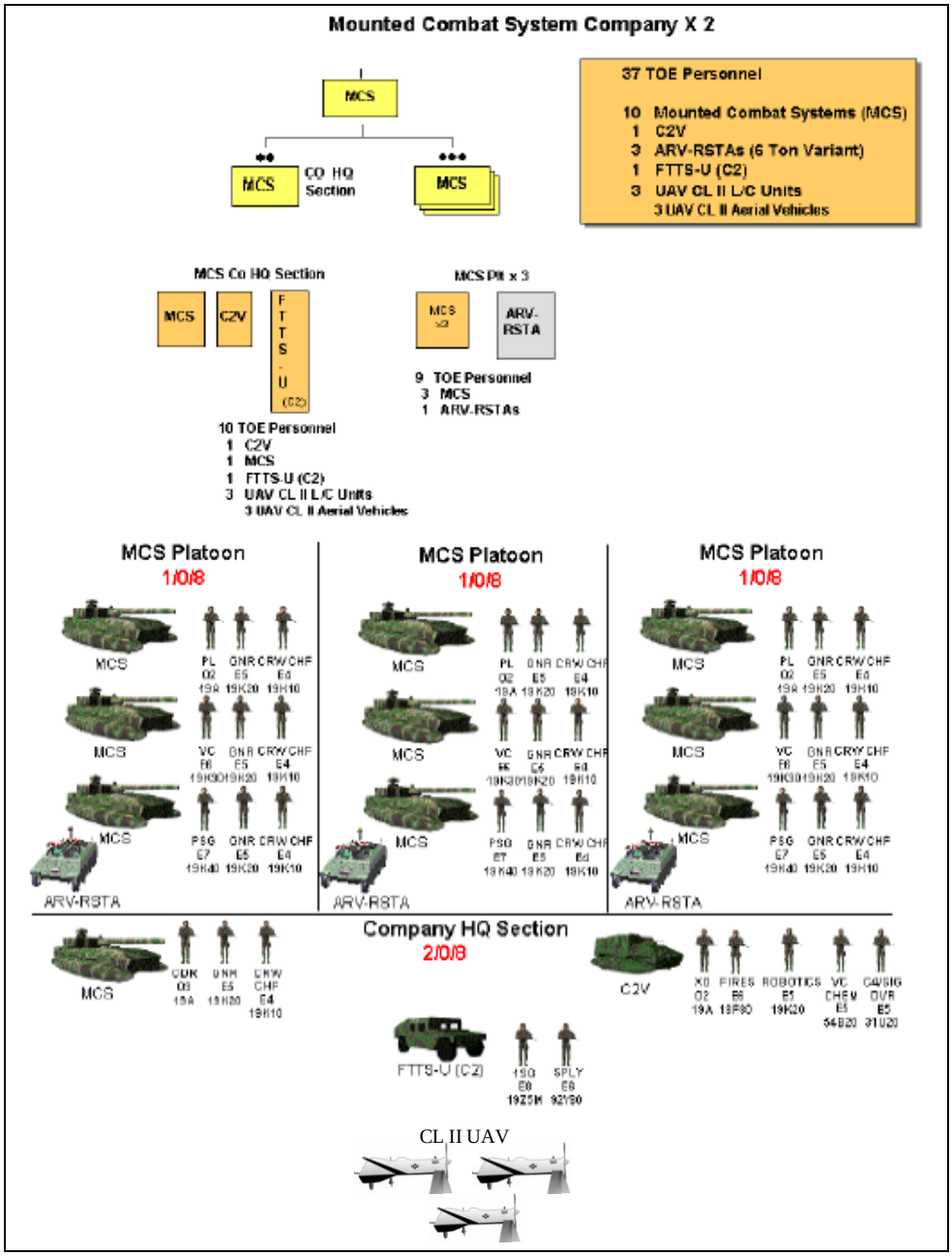


Figure B-1. Unit organizational chart for the MCS.

INTENTIONALLY LEFT BLANK.

Appendix C. Mission Decomposition and Product Development

The mission decomposition process and product development for the demonstration were executed in several phases: research and analysis followed by the development of a tactical scenario, courses of action (COAs), execution matrix, and mission threads.

C.1 Research and Analysis

Research was conducted on Future Combat System (FCS) platforms and components. We focused on the Mounted Combat System (MCS) Company with support elements from the Unit of Action (UA) and Combined Arms Battalion (CAB), especially the Non-Line-of-Sight Cannon (NLOS-C) Battalion. The characteristics and components of threat-mounted and dismounted infantry forces were also researched.[1–3]

C.2 Tactical Scenario Development

The development process began by selecting an existing Unit of Action Maneuver Battle Lab scenario to build the "Road to War" and create the tactical environment for the demonstration. The Road to War outlines the area of operations and sequence of events that led to a situation (or conflict) at a specific time period. For the demonstration, the Road to War begins in the year 2015 and takes place in the fictitious countries of Orangeland and Blueland. The year 2015 was chosen for the scenario so that FCS ORD and Organization and Operation (O&O) objective specifications could be used for analysis. The fictitious countries overlay the terrain in and around Fort Knox, Kentucky.

The Road to War was derived from the Caspian Sea scenario, mounted exploitation/pursuit operations (modified Fort Knox version), phase 3 of vignette 3. The Road to War outlined the mission of a combined joint task force (CJTF) that included a combined forces land component command, a unit of employment (UE1), two units of action (UA1 and UA2) and their associated combined arms battalions (CAB1 and CAB2), and MCS and infantry companies. Using the UA1, we scaled down the tactical scenario to the CAB, MCS company level, specifically CAB2 and MCS Company A. An NLOS-C battery also supported the MCS company. Task organization for the CAB and MCS Company was based on TRADOC PAM 525-3-90.[4]

---

[1]Deitz, P. H.; Starks, M. W. *The Generation, Use, and Misuse of "PKs" in Vulnerability/Lethality Analyses*; ARL-TR-1640; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, March 1998.

[2]Apicella, F. J.; Wyan, K. W.; Wilcox, C. M. ATEC Initiatives in Response to the Office of the Secretary of Defense Policy Guidelines for Test and Evaluation. *International Test and Evaluation Association (ITEA) Journal* **2009**, *30*, 361–368.

[3]Operational Requirements Document for the Future Combat Systems (ORD) UAMBL Fort Knox, KY, version 3, 14 April 2003.

[4]Headquarters, Department of the Army. *O&O Plan Maneuver Unit of Action*; TRADOC PAM 525-390; Fort Monroe, VA, 30 June 2003.

C.2.1 UA and CAB Road to War

In January 2015, an extreme fundamentalist group stages a coup and ousts the democratically elected government of Orangeland. The ousted government flees in exile to the southern country of Blueland. By March 2015, the UN passes a resolution to restore the exiled government by force. A CJTF is formed and shortly thereafter, Orangeland forces attack south, taking over two-thirds of Blueland. The CJTF's mission is to deploy to Blueland to drive out rebel forces, restore the rightful government of Orangeland, and eliminate the weapons of mass destruction threat.

At "H" hour, U.S. forces deploy to Blueland and conduct entry operations into the port of Mobile. The demonstration vignette begins at H+85 when rebel forces have been driven north out of Blueland, and the UE1 has received a change of mission because of a change in rebel posture. At H+86, UA1 has occupied positions in staging area (SA) Elizabethtown and receives their mission order to attack north and seize "Objective Camel." UA1 provides change-of-mission orders to each element of the UA. At H+86, CAB2 is tasked to conduct the mission described in figure C-1. At H+88, UA1 is in SA Elizabethtown preparing for attack. Between H+88 and H+90, MCS Company A (MCS A) moves into attack position (AP) Muldraugh and conducts pre-combat checks. The actual demonstration events take place between H+90 and H+98.

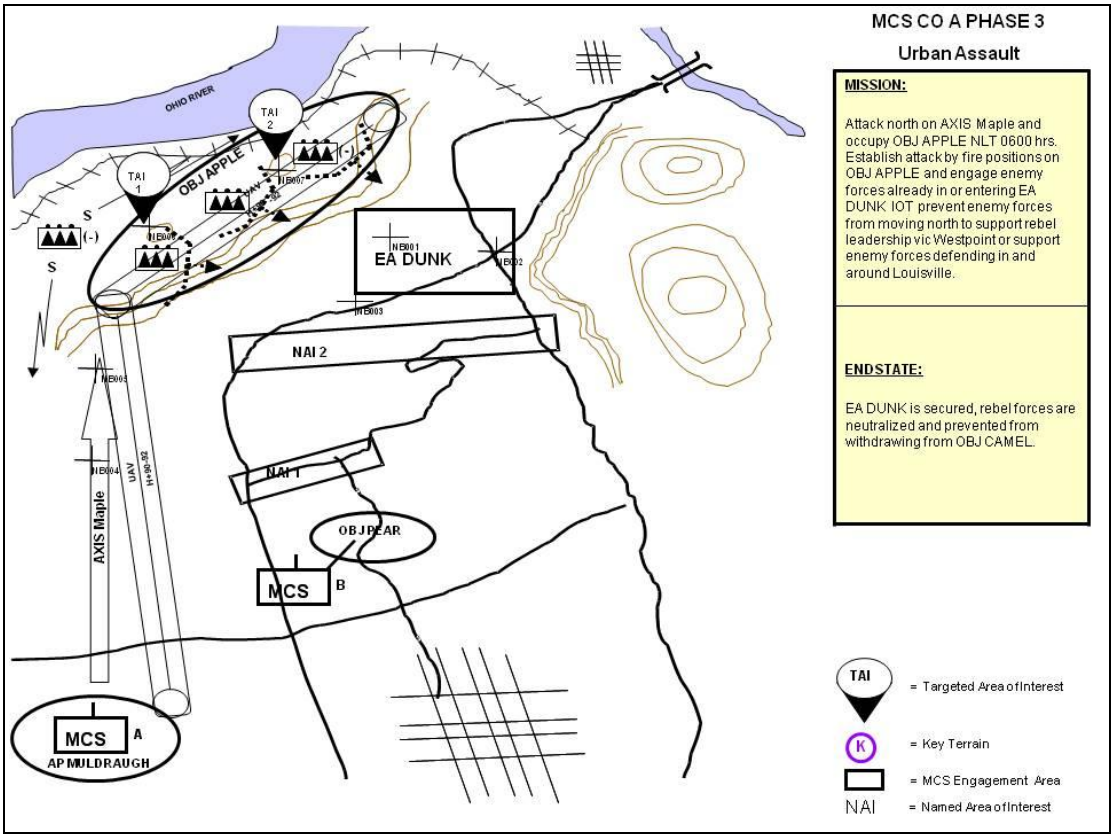


Figure C-1. Mission statement for MCS A.

C.2.2 Vignette Specifications

The particular vignette developed for modeling was a movement by an MCS company to occupy an objective and establish an "attack by fire" position overlooking engagement area "Dunk." The NLOS-C battery provided protective fire for movement. The surveillance assets included a suite of sensors mounted on unmanned aerial vehicles (UAVs) and armored robotic vehicles (ARVs). The UAV and ARV were controlled by the company headquarters section using the command and control vehicle (C2V) capabilities. The C2V-mounted headquarters section made the decisions to engage targets based on the sensor feeds; it called for fires from the available NLOS-Cs (that is, from those that had the capability to engage the targets and were not already busy conducting other fires). The selected NLOS-C platforms executed the fire missions using a hypothetical smart munition, and the resulting damage state of the target(s) was updated. A backup for the C2V was modeled to take over control should the C2V be disabled.

The opposing force consisted of six squad- or platoon-sized units that were quite ineffective in most replications of the vignette.

C.3 COA Development

Once the MCS-A mission was defined, COAs were developed. A COA is a possible solution to achieve the unit's mission that complies with the commander's guidance and intent. For the MMF demonstration, the focus was on the C2V, ARV, UAV, and NLOS-C platforms. The COA was developed in three phases: reconnaissance, movement, and operations on the objective. The COA statement and sketch for MCS A is shown in figure C-1.

War-gaming is then used to determine alternate COAs for tactical planning. The war-gaming process is disciplined,[5] relying on rules and steps that help Warfighters to visualize the flow of a battle. War-gaming stimulates ideas, provides insights that might not otherwise be discovered, and highlights critical tasks and the possible outcomes of military actions that could affect mission success or failure. The process considers friendly and enemy strengths and weaknesses, resources, COAs, and characteristics of the environment. When attention is focused on each phase of the operation in a logical sequence, the process relies heavily on the doctrinal foundation, tactical judgment, and experience of the war-gaming staff. It is an iterative process that documents an action, reaction, and counteraction.

In preparation for the war game, tools were identified and a list of rules with assumptions was constructed. The following tools are included in war-gaming:

- Maps and overlays for the area of operation.
- Symbols of both friendly and enemy assets that will conduct the operation.

[5]Headquarters, Department of the Army. The Military Decision-Making Process. In *Staff Organization and Operations*; FM 101-5; Washington, DC, 31 May 1997.

- A list of critical tasks.
- Key decision points.
- Assumptions and evaluation criteria normally derived from the "Commander's Intent."

The general rules of war-gaming are as follows:

1. Remain objective about an interaction, outcome, or response.
2. Select a method to accurately record the results of the war game.
3. Avoid drawing premature conclusions.

C.4 Execution Matrix

The MCS A mission was broken into three phases during COA development: maneuver to Objective Apple, establish attack by fire positions, and engage forces in EA DUNK. An execution matrix was developed to depict the actions of MCS A over the 8-h period (shown in table C-1). The matrix was organized by phase and battlefield operating system beginning at H+90 (0200 h), when MCS A is occupying AP Muldraugh and preparing for future operations.

Table C-1. Execution matrix for MCS company (8-h period).

Time	**H+90 Phase I Reconnaissance 0200 H**	**H+92 Phase II Movement**	**H+94 Phase III Decisive Operations on Objective**	**H+96**	**H+98** → **1000 H**
Threat	Suspected enemy occupation.	Enemy recon on hilltops (key terrain).	Enemy engaging Blue force in city.	Enemy attempting to escape north of city on key routes.	Enemy withdrawn from OBJ.
Intelligence/ Recon	UAV recon axis and OBJ area. Report movements and enemy activity.	ARV-RSTAs recon Axis Maple. UAVs survey OBJ Apple.	UAV recon EA DUNK and NAIs. ARV-RSTA surveillance of EA DUNK, flank second.	UAV report enemy locations and movement at NAI.	UAV conducting battle damage assessment (BDA). Report BDA.
Maneuver	UAV cross SP.	ARVs cross SP first then platoons cross SP and move to OBJ Apple.	Occupy OBJ Apple O&O attack by fire EA DUNK.	Attack to destroy enemy forces EA DUNK.	EA DUNK secured.
Fire Support	—	Provide fire support for move.	On call fires EA DUNK.	On call fires EA DUNK.	Fire missions O&O.
C2	Establish communications.	Receive and report intelligence. Track MCS-A movement.	Receive and report intelligence information to battle management.	Receive and report BDA information to battle management.	Receive and report BDA to battle management.
NAI/TAI	TAI 1, TAI 2	TAI 1, TAI 2	NAI 1, NAI 2	NAI 1, NAI 2	NAI 2
CCIR decision point	Does the enemy occupy OBJ Apple?	Where are enemy obstacles along routes?	What are the enemy location/movement routes?	Which routes will the enemy use to withdraw from city?	Where are enemy remnants, if any?

C.4.1 Phase I (H+90 Through H+92) Reconnaissance

During this 2-h reconnaissance phase, MCS A conducts reconnaissance operations along AXIS Maple to identify enemy forces that could affect their movement up to Objective Apple. The reconnaissance phase is critical for MCS A to gain SA and set the conditions for beginning phase II.

C.4.2 Phase II (H+92 Through H+94) Movement

During phase II, MCS A begins movement north out of AP Muldraugh toward Objective Apple. This 2-h time window is the most critical period of the operation for the company. MCS A is exposed to the greatest risk of encountering enemy forces or being attacked. For demonstration purposes, enemy activity was primarily scripted to occur during phase II. As MCS A moves north along AXIS MAPLE, the company receives enemy artillery fire. The effect of the artillery fire is the loss of digital communications in the C2V. The enemy interaction occurs at 0524 h before the MCS Company has reached Objective Apple. This event initiates the effort to trace the mission thread from the MMF level 1, Interaction (MCS A receives enemy artillery fire), up through level 7, Mission Accomplishment (occupy Objective Apple no later than 0600 h and establish attack by fire positions to engage enemy forces in EA DUNK). The impact of the enemy interaction during phase II is critical since the mission requires the company to occupy Objective Apple no later than 0600 h. The commander has no more than fifteen minutes to make an appropriate decision and act on it. The degradation of the C2V platform as a result of this enemy interaction was then the focus for the remainder analysis and formed the basis for the development of alternative COAs.

C.4.3 Phase III (H+94 Through H+1000) Decisive Operations and Objective

Three alternative COAs were constructed for the MMF demonstration. These alternatives were in response to the loss of digital communications for the C2V caused by a ballistic interaction. The COAs traced task performance from platform task to collective task to mission accomplishment. The COAs were designed to overcome the effect of the C2V degradation on essential collective tasks by transferring functions to alternative platforms available to MCS A through the system of systems. The transfer of function brings task performance back up to an acceptable level for the MCS A to continue its mission.

The military decision-making process (MDMP) used to assess risk and adjust resources is also reflected in the mission thread using the Army Universal Task List, “Conduct Risk Assessment” and “Adjust Resources.” The COA developed specifically identified alternate methods to establish SA on and around the objective (shown in table C-2). Each COA was scripted onto a time-ordered event list (TOEL) and documented in Microsoft Project for model input.

Table C-2. Alternate courses of action.

COA	Action	Outcome
COA-1	Transfer control of UAVs to 1st and 2nd platoons: Orders C2V to transfer control of UAVs to 1st and 2nd platoons. Takes control of SA/fires. Orders company to continue advance to Objective Apple (5 km/h).	30-min delay to transfer operational control of UAVs and assume SA/fires control.
COA-2	Transfer control of UAVs to future tactical truck system (FTTS): Take control of fires. Fire direction noncommissioned officer transfers to commander's vehicle to control fires. SA and fire control transferred to FTTS. Orders C2V to transfer control of UAV 1 and 2 to FTTS. Robotics noncommissioned officer transfers to FTTS. Orders launch and recovery equipment transferred to 2nd platoon. The First Sergeant (1SG) transfers to 3rd platoon security force. Requests Contact Maintenance Team for Battalion trains meet company on Objective Apple to repair C2V digital communications. Orders company to resume advance toward Objective Apple at increased speed (10 km/h).	15-min delay to transfer operational control of UAVs to FTTS and to assume SA/fires control. Delay offset by increased speed.
COA-3	Request support from cab to pick up feed from UAVs 1 and 2: Orders CAB to pick up feeds from UAVs 1 and 2 and to send updated feeds to the MCS Commander C2V to transfer control of UAVs to 1st and 2nd platoons. Takes control of SA/fires. Orders company to continue advance to Objective Apple (5 km/h).	15-min delay while CAB assumes control of UAVs 1 and 2 and MCS Commander assumes SA/fires control.

The important difference between the COAs is the time necessary to implement. COAs 2 and 3 take 15 min to implement, but COA 1 takes 30 min to implement. Given the tactical situation at the time the ballistic interaction occurred, a delay of more than 15 min could lead to mission failure. Therefore, choosing COA 1 will likely lead to mission failure, whereas COAs 2 and 3 will lead to mission accomplishment.

C.5 Mission Thread Development

A step-by-step analysis was conducted of the operation, phase by phase, using the action, reaction, and counteraction MDMP. The step-by-step analysis identified the sequence of tasks to be performed. By using the Joint Training Information Management tool, the analysis also determined the purpose, measures of effectiveness (MOEs), and measures of performance (MOPs) for each platform task.

To make the vignette into a script that could be modeled as a sequence of tasks, a TOEL was developed that would happen (or might happen) in the vignette. The list was then further refined into lower-level tasks that each platform (or small unit) would execute as a function of time or situation to accomplish the mission. Task attributes included duration, triggers for start and stop, dependencies, and interrelationships. The mission thread, as developed via the MMF framework, ultimately represents a task-based fault tree to translate (1) the effect of component state changes on the task(s) being performed by the platform/system, (2) the impact of platform-level task failures on supported collective tasks, and (3) the impact of essential collective task failures on the mission.

Similar to the scripted COAs, Microsoft Project was used to organize and display the FCS platform TOEL. While some assumptions were made to determine time intervals for task performance, the assumptions were based on doctrine as well as military experience and were considered adequate for planning purposes. Once populated, the Microsoft Project Gantt view provided the ability to analyze a cross section of platforms performing tasks at any given time. The utility of this view became apparent as enemy interactions were scripted to degrade platform capabilities. These degradations could be traced from platform to task performance by drawing a vertical line from the interaction down the chart and are what triggered the changes in the tactical scenario. The notes page of the Microsoft Project file provided a convenient location to document associated task information that included purpose, conditions, MOE, and MOP for each task. The information was tailored to capture all aspects of task performance in this specific mission thread.

INTENTIONALLY LEFT BLANK.

Appendix D. Platform Functional Definitions

The U.S. Army Research Laboratory defined platform functions (level 3) by systems engineering analysis. The product of this analysis was the identification of critical components and their relationship to individual platform functional capabilities.

Fault tree representation was then used to transform level-3 functions into a format that could be used in the model. A fault tree is a graphical representation of a Boolean expression. The expression is built up by using Boolean ANDs and ORs to join together names of subsystems and components. It expresses the nature of the dependence of some functionality on the condition of those subsystems and components. The fault tree may be seen as a road network between two terminal nodes. From this perspective, if any of the subsystems or components are nonfunctional, they are deleted, which at least partially severs the network. The fault tree asserts that the corresponding functionality is available exactly when the network retains at least one path that connects the two terminal nodes. Each subsystem in a fault tree may, in general, have its own expansion as a separate fault tree. Figure D-1 is an example of the fault tree representation for delivery accuracy used in the demonstration. If the transmitter of the projectile tracking system is nonfunctional, the delivery accuracy of the system would be reduced by 25%. However, if only one receiver is lost and nothing else, delivery accuracy should still be intact.

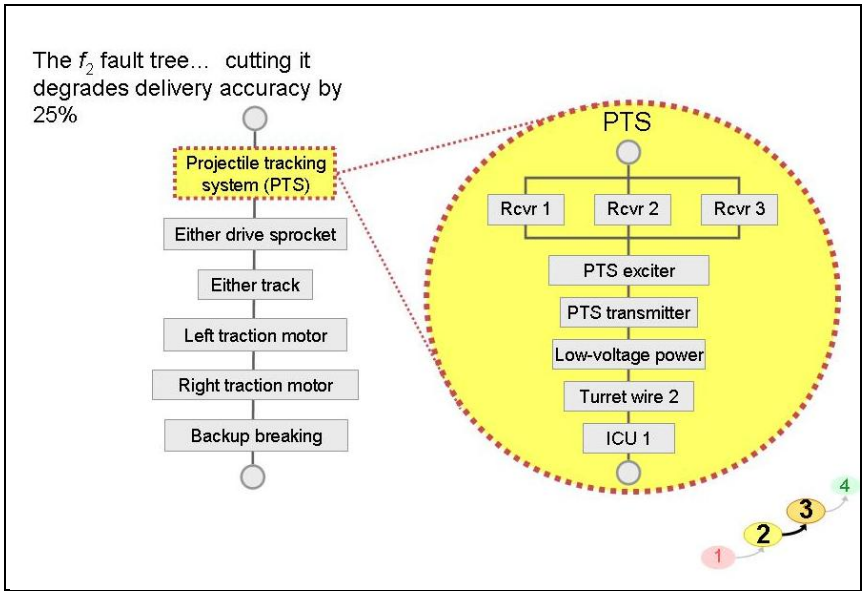


Figure D-1. Fault tree representation.

The level-3 capabilities were then organized, or binned, into seven categories: mobility, firepower, communications, target acquisition, protection, surveillance, and other. The capabilities were further defined within a collection of ~50 elements of capability degradation, as shown in figure D-2.

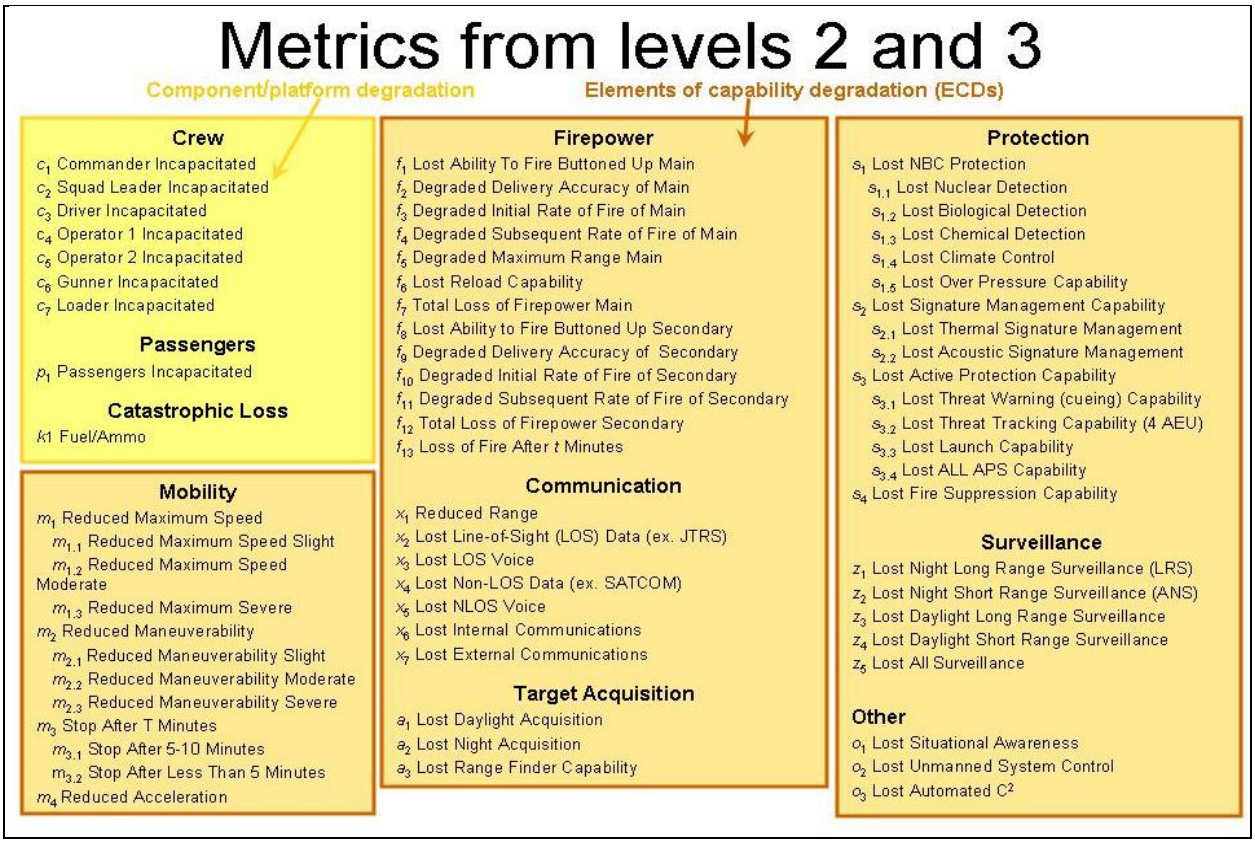


Figure D-2. Elements of capability degradation.

The ECD collection for the demonstration is shown in table D-1. The fault tree representation of the ECD then follows. Typically only a subset of the states applies to any particular platform type; however, the same numbering scheme was used for all platform types. Functional descriptions that did not apply to a type were simply not used in that case. Also, this list has been modified several times since the first SBM demonstration, so other sources may include more or different functional descriptions.

Table D-1. Elements of capability degradation for the MMF demonstration.

Mobility	M1	Reduced forward speed
	M2	Reduced maneuverability
	M3	Stop after time T
	M4	Total immobilization
Firepower	F1	Lost ability to fire buttoned up
	F2	Degraded delivery accuracy
	F3	Degraded initial rate of fire (time to fire first round)
	F4	Degraded subsequent rate of fire
	F5	Total loss of firepower
Target Acquisition	A1	Weapon day sight lost
	A2	Weapon night sight lost
Surveillance	Z1	Loss of primary sensor
	Z2	Loss of secondary sensor
	Z3	Loss of tertiary sensor
	Z4	Loss of vision block
Communications	X1	Loss of external data communications
	X2	Loss of external voice communications
	X3	Loss of internal communications
	X4	Loss of LAN (local area network)
	X5	Loss of all communications
Survivability	S1	Loss of NBC (nuclear biological chemical) protection
	S2	Loss of (ability to deploy) obscurants
	S3	Loss of silent watch capability
	S4	Loss of APS (active protection system)
	S5	Loss of secondary armament
Crew	C1	Commander incapacitated
	C2	Squad leader incapacitated
	C3	Driver incapacitated
	C4	Operator 1 incapacitated
	C5	Operator 2 incapacitated
	C6	Gunner incapacitated
	C7	Loader incapacitated
Other_State_Effects	O1	Loss of situation awareness
Catastrophic	K1	Catastrophic kill (all capabilities lost)

The following definitions are the top-level capability state vectors played for the command and control vehicle (C2V):

Mobility
CV_m1_Reduced_Maximum_Speed,
CV_m2_Reduced_Maneuverability,
CV_m3_Stop_After_Time_t,
CV_m4_Total_Immobilization,
Firepower
CV_f1_lethality_of_CV_remote_weapon_system,
CV_f2_lethality_of_CV_buttoned_up,

```
# Target Acquisition
        CV_a1_primary_weapon_sight,
        CV_a2_day_vision,
        CV_a3_night_vision,
        CV_a4_daylight_sights_2_2000M,
        CV_a5_night_sights_2_1500M,
# Surveillance
        CV_z1_vision_blocks,
# Communications
        CV_x1_external_voice_communication,
        CV_x2_external_data_communication,
        CV_x3_internal_communication,
        CV_x4_all_communications,
        CV_x5_LAN_communications,
# Survivability
        CV_s1_nbc_protection,
        CV_s2_obscurance,
        CV_s3_silent_watch,
# Crew
        CV_c1_commander_incapacitated,
        CV_c2_squadleader_incapacitated,
        CV_c3_driver_incapacitated,
        CV_c4_both_operators,
# Other_Mission_Functions
        CV_o1_situational_awareness,
# Catastrophic
        CV_k1_catastrophic_loss
```

The following system fault trees are in MUVES format for Boolean operations where an "&" represents the "AND" and "|" represents "OR" for the C2V. The fault trees define the relationship of components to system functional capability.

```
CV_m1_Reduced_Maximum_Speed =
        loss_pwr_to_2wheels_CV1 |
        loss_pwr_to_4wheels_CV1 |
        one_flat_tire_CV1       |
        two_flat_1n1_tires_CV1 |
        two_flat_2n0_tires_CV1 |
        three_flat_2n1_tires_CV1 |
        three_flat_3n0_tires_CV1 |
        four_flat_2n2_tires_CV1 |
        four_flat_3n1_tires_CV1 |
        four_flat_4n0_tires_CV1

CV_m2_Reduced_Maneuverability =
         normal_shifting_pk_CV1 |
```

```
        power_steering_pk_CV1 |
        front_steering_pk_CV1 |
        rear_steering_pk_CV1          |
        service_brakes_pk_CV1         |
        one_leading_wheel_CV1 |
        no_data_sys_pk_CV1 |
        all_gps_pk_CV1        |
        loss_eplrs_pk_CV1

CV_m3_Stop_After_Time_t =
        main_fuel_cells_pk_CV1

CV_m4_Total_Immobilization =
        loss_pwr_to_6wheels_CV1 |
        loss_pwr_to_8wheels_CV1 |
        two_opp_wheels_CV1          |
        two_wheels_2n0_CV1          |
        five_flat_2n3_tires_CV1 |
        five_flat_4n1_tires_CV1 |
        six_flat_3n3_tires_CV1        |
        six_flat_4n2_tires_CV1        |
        seven_flat_tires_CV1 |
        eight_flat_tires_CV1  |
        engine_pk_CV1                 |
        transmission_pk_CV1           |
        all_steering_pk_CV1  |
        all_throttle_control_pk_CV1 |
        main_power_pk_CV1

CV_f1_lethality_of_CV_remote_weapon_system =
        rws_weapon_sys_CV1

CV_f2_lethality_of_CV_buttoned_up =
        remote_weapon_station_sys_CV1

#Acquisition
CV_a1_primary_weapon_sight =
        all_prim_weap_sights_pk_CV1

CV_a2_day_vision =
        driver_vision_pk_CV1 &
        commander_vision_pk_CV1 &
        driver_night_vis_pk_CV1

CV_a3_night_vision =
        commander_night_vis_pk_CV1
```

```
CV_a4_daylight_sights_2_2000M =
        rws_day_sight_sys_CV1

CV_a5_night_sights_2_1500M =
        rws_night_sight_sys_CV1

# Surveillance
CV_z1_vision_blocks =
        driver_vision_pk_CV1 |
        commander_vision_pk_CV1 |
        driver_night_vis_pk_CV1 |
        commander_night_vis_pk_CV1

#Communications
CV_x1_external_voice_communication =
        ext_voice_comm_sys_CV1

CV_x2_external_data_communication =
        all_digital_commo_pk_CV1

CV_x3_internal_communication =
        all_internal_commo_pk_CV1|
        driver_intcom_pk_CV1                |
        commander_intcom_pk_CV1         |
        squadleader_intcom_pk_CV1

CV_x4_all_communications =
        all_communications_pk_CV1

CV_x5_LAN_communications =
        sep_pk_CV1                      |
        inverter_pwr_pk_CV1

#Survivability
CV_s1_nbc_protection =
        nbc_protection_sys_pk_CV1
CV_s2_obscurance =
        smoke_obscur_sys_pk_CV1
CV_s3_silent_watch =
        auxiliary_power_sys_CV1

#Crew
CV_c1_commander_incapacitated =
        commander_only_CV1
CV_c2_squadleader_incapacitated =
        squadleader_only_CV1
```

```
CV_c3_driver_incapacitated =
        driver_only_CV1
CV_c4_both_operators = both_operators_CV1

#Other Mission Functions
CV_o1_situational_awareness =
        fbcb2_sys_CV1 |
        asas_only_pk_CV1 |
        sotm_only_pk_CV1

# Catastrophic
CV_k1_catastrophic_loss =
        CV_catastrophic_fuel |
        CV_catastrophic_ammo
```

The following definitions are the top-level capability state vectors played for the reconnaissance vehicle:

```
# Mobility Capabilities
        RSV1_m1_Reduced_Maximum_Speed,
        RSV1_m2_Reduced_Maneuverability,
        RSV1_m3_Stop_After_Time_t,
        RSV1_m4_Total_Immobilization,
# Firepower Capabilities
        RSV1_f1_lethality_of_RSV1_remote_weapon_system,
        RSV1_f2_lethality_of_RSV1_buttoned_up,
# Target Acquisition
        RSV1_a1_LongRangeAcquisiton,
        RSV1_a2_day_vision,
        RSV1_a3_night_vision,
        RSV1_a4_daylight_sights_2_2000M,
        RSV1_a5_night_sights_2_1500M,
# Surveillance
        RSV1_z1_vision_blocks,
# Communications
        RSV1_x1_external_voice_communication,
        RSV1_x2_external_data_communication,
        RSV1_x3_internal_communication,
        RSV1_x4_all_communications,
# Survivability
        RSV1_s1_nbc_protection,
        RSV1_s2_obscurance,
        RSV1_s3_silent_watch,
# RSV Crew
        RSV1_c1_commander_incapacitated,
        RSV1_c2_squadleader_incapacitated,
```

```
        RSV1_c3_driver_incapacitated,
        RSV1_c4_both_operators,
# Other Mission Functions
        RSV1_o1_situational_awareness,
# Catastrophic
        RSV1_k1_catastrophic_loss
```

The following system fault trees represent the reconnaissance vehicle functional capabilities:

```
RSV1_m1_Reduced_Maximum_Speed =
        loss_pwr_to_2wheels_RSV1 |
        loss_pwr_to_4wheels_RSV1 |
        one_flat_tire_RSV1    |
        two_flat_1n1_tires_RSV1 |
        two_flat_2n0_tires_RSV1 |
        three_flat_2n1_tires_RSV1 |
        three_flat_3n0_tires_RSV1 |
        four_flat_2n2_tires_RSV1 |
        four_flat_3n1_tires_RSV1 |
        four_flat_4n0_tires_RSV1

RSV1_m2_Reduced_Maneuverability =
        normal_shifting_pk_RSV1 |
        power_steering_pk_RSV1 |
        front_steering_pk_RSV1 |
        rear_steering_pk_RSV1       |
        service_brakes_pk_RSV1      |
        one_leading_wheel_RSV1 |
        no_data_sys_pk_RSV1         |
        all_gps_pk_RSV1      |
        loss_eplrs_pk_RSV1

RSV1_m3_Stop_After_Time_t =
        main_fuel_cells_pk_RSV1

RSV1_m4_Total_Immobilization =
        loss_pwr_to_6wheels_RSV1 |
        loss_pwr_to_8wheels_RSV1 |
        two_opp_wheels_RSV1         |
        two_wheels_2n0_RSV1         |
        five_flat_2n3_tires_RSV1 |
        five_flat_4n1_tires_RSV1 |
        six_flat_3n3_tires_RSV1       |
        six_flat_4n2_tires_RSV1       |
        seven_flat_tires_RSV1         |
        eight_flat_tires_RSV1         |
```

```
        engine_pk_RSV1              |
        transmission_pk_RSV1        |
        all_steering_pk_RSV1        |
        all_throttle_control_pk_RSV1 |
        main_power_pk_RSV1

# Firepower
RSV1_f1_lethality_of_RSV1_remote_weapon_system =
        rws_weapon_sys_RSV1

RSV1_f2_lethality_of_RSV1_buttoned_up =
        remote_weapon_station_sys_RSV1


# Acquisition
RSV1_a1_LongRangeAcquisiton =
        LRAS3          |
        Cupola|
        (MissionSpecialist1 & MissionSpecialist2)

RSV1_a2_day_vision =
+
        driver_vision_pk_RSV1 &
        commander_vision_pk_RSV1 &
        driver_night_vis_pk_RSV1

RSV1_a3_night_vision =
        commander_night_vis_pk_RSV1

RSV1_a4_daylight_sights_2_2000M =
        rws_day_sight_sys_RSV1

RSV1_a5_night_sights_2_1500M =
        rws_night_sight_sys_RSV1

# Surveillance
RSV1_z1_vision_blocks =
        driver_vision_pk_RSV1 |
        commander_vision_pk_RSV1 |
        driver_night_vis_pk_RSV1 |
        commander_night_vis_pk_RSV1

RSV1_z2_LongRangeAdvancedScout =
        LRAS3          |
        Cupola|
        (MissionSpecialist1 & MissionSpecialist2)
```

```
# Communications
RSV1_x1_external_voice_communication =
        ext_voice_comm_sys_RSV1

RSV1_x2_external_data_communication =
        all_digital_commo_pk_RSV1

RSV1_x3_internal_communication =
        all_internal_commo_pk_RSV1      |
        driver_intcom_pk_RSV1           |
        commander_intcom_pk_RSV1        |
        squadleader_intcom_pk_RSV1

RSV1_x4_all_communications =
        all_communications_pk_RSV1

# Survivability
RSV1_s1_nbc_protection =
        nbc_protection_sys_pk_RSV1
RSV1_s2_obscurance =
        smoke_obscur_sys_pk_RSV1
RSV1_s3_silent_watch =
        auxiliary_power_sys_RSV1

# Crew
RSV1_c1_commander_incapacitated =
        commander_only_RSV1
RSV1_c2_squadleader_incapacitated =
        squadleader_only_RSV1
RSV1_c3_driver_incapacitated =
        driver_only_RSV1
RSV1_c4_both_operators =
        both_operators_RSV1

# Other Mission Functions
RSV1_o1_situational_awareness =
        fbcb2_sys_RSV1 |
        asas_only_pk_RSV1 |
        sotm_only_pk_RSV1 |
        printer_only_pk_RSV1

# Catastrophic
RSV1_k1_catastrophic_loss=
        RSV1_catastrophic_fuel |
        RSV1_catastropic_ammo
```

The following definitions are the top-level capability state vectors played for the non-line-of-sight cannon:

Mobility =
NLOS1_m1_reduced_maximum_speed |
NLOS1_m2_reduced_maneuverability |
NLOS1_m3_stop_after_time_t |
NLOS1_m4_total_immobilization

Firepower =
NLOS1_f2_reduced_delivery_accuracy_of_main_armament |
NLOS1_f3_reduced_initial_rate_of_fire_of_main_armament |
NLOS1_f4_reduced_subsequent_rate_of_fire_of_main_armament |
NLOS1_f5_main_armament

Communicate =
NLOS1_x1_reduced_range_of_external_communication |
NLOS1_x2_loss_of_external_voice_communication

Survivability =
NLOS1_s1_secondary_armament

Situational_Awareness =
NLOS1_o1_situational_awareness

Catastrophic =
NLOS1_k1_catastrophic_ammo |
NLOS1_k2_catastrophic_fuel

The following system fault trees represent the Non-Line-of-Sight Cannon (NLOS-C) functional capabilities:

NLOS1_m1_reduced_maximum_speed =
turbocharger_system_NLOS1|
supercharger_system_NLOS1 |
road_wheel_left_1_system_NLOS1 |
road_wheel_right_1_system_NLOS1|
track_tensioner_left_system_NLOS1|
track_tensioner_right_system_NLOS1 |
shock_absorber_left_front_system_NLOS1 |
shock_absorber_left_rear_system_NLOS1 |
shock_absorber_right_front_system_NLOS1|
shock_absorber_right_rear_system_NLOS1

NLOS1_m2_reduced_maneuverability =
shock_absorber_left_front_system_NLOS1 |

```
        shock_absorber_left_rear_system_NLOS1   |
        shock_absorber_right_front_system_NLOS1|
        shock_absorber_right_rear_system_NLOS1 |
        road_wheel_left_1_system_NLOS1  |
        road_wheel_left_2_system_NLOS1  |
        road_wheel_left_3_system_NLOS1  |
        road_wheel_left_4_system_NLOS1  |
        road_wheel_left_5_system_NLOS1  |
        road_wheel_left_6_system_NLOS1  |
        road_wheel_left_7_system_NLOS1  |
        road_wheel_right_1_system_NLOS1|
        road_wheel_right_2_system_NLOS1|
        road_wheel_right_3_system_NLOS1|
        road_wheel_right_4_system_NLOS1|
        road_wheel_right_5_system_NLOS1|
        road_wheel_right_6_system_NLOS1|
        road_wheel_right_7_system_NLOS1|
        track_tensioner_left_system_NLOS1|
        track_tensioner_right_system_NLOS1        |
        brakes_NLOS1

NLOS1_m3_stop_after_time_t =
        lubrication_system_NLOS1   |
        cooling_system_NLOS1

NLOS1_m4_total_immobilization =
        idler_wheel_left_system_NLOS1     |
        idler_wheel_right_system_NLOS1    |
        final_drive_sprocket_left_system_NLOS1    |
        final_drive_sprocket_right_system_NLOS1 |
        left_track_system_NLOS1     |
        right_track_system_NLOS1  |
        transmission_system_NLOS1|
        transfer_unit_system_NLOS1|
        driver_controls_system_NLOS1       |
        fuel_system_NLOS1  |
        engine_system_NLOS1

#Lethality
NLOS1_f2_reduced_delivery_accuracy_of_main_armament =
        automatic_fire_control_data_system_NLOS1
NLOS1_f3_reduced_initial_rate_of_fire_of_main_armament =
        rammer_system_NLOS1       |
        automatic_fire_control_complete_system_NLOS1   |
        dynamic_reference_unit_power_system_NLOS1     |
        vms_system_NLOS1
```

```
NLOS1_f4_reduced_subsequent_rate_of_fire_of_main_armament =
        rammer_system_NLOS1       |
        (brakes_NLOS1          &
          spade_NLOS1         )       |
        automatic_fire_control_complete_system_NLOS1
NLOS1_f5_main_armament =
        gun_mount_system_NLOS1  |
        cannon_system_NLOS1       |
        elevation_equilibration_mechanism_system_NLOS1          |
        manual_traverse_system_NLOS1

# Communications
NLOS1_x1_reduced_range_of_external_communication =
        reduced_range_of_external_communication_NLOS1

NLOS1_x2_loss_of_external_voice_communication =
        voice_radio_system_NLOS1  &
        digital_radio_system_NLOS1

# Survivability
NLOS1_s1_secondary_armament =
        50_caliber_machine_gun_NLOS1

# Crew
NLOS1_c1_commander_incapacitated =
        chief_of_section_NLOS1

NLOS1_c2_gunner_incapacitated =
        gunner_NLOS1

NLOS1_c3_driver_incapacitated =
        driver_NLOS1

NLOS1_c4_loader_incapacitated =
        loader_NLOS1

# Other_Mission_Functions
NLOS1_o1_situational_awareness =
        fbcb2_NLOS1 |
        asas_NLOS1

# Catastropic
NLOS1_k1_catastrophic_ammo =
        propellant_NLOS1 |
        projectile_NLOS1
NLOS1_k2_catastrophic_fuel =
        upper_fuel_tank_NLOS1 |
        lower_fuel_tank_NLOS1
```

INTENTIONALLY LEFT BLANK.

Appendix E. Platform Representation and Model Object Classes

There were four platform representations in the Storyboard Model (SBM): movers, sensors, communicators, and shooters.

The representation of the own force (OWNFOR) at the platform level consisted of assigning each platform its own set of initial (undegraded) capabilities.* These capabilities were then modified as events occurred in the simulation.† The task workload also changed as events occurred. At each event, the modified capabilities were compared to the current platform tasks and capability requirements for task execution. With this approach, the SBM was able to determine whether platforms had sufficient capabilities to continue the mission objective.

Processors were developed to generate platform capability state changes (combat damage, failures, and repairs) and merged with the schedule of platform-level tasks to form a script of externally generated events.‡ There were, however, some random events dynamically modeled within the SBM that are also described in appendix F.

E.1 Movers

All platforms were potentially movers (those that remained stationary had a single waypoint) and had a path with associated functions and data attributes. A path consisted of an array of the nodes (waypoints) to be visited in sequence. Each node had four associated data values: x-coordinate (east-west), y-coordinate (north-south), a dwell time at that node, and a desired departure speed (the maximum speed at which the platform would depart from that node when the dwell time had expired). With a few exceptions, the actions of the movers can be described as follows:

1. Begin at the first waypoint (part of the initialization process at the start of the vignette).
2. Stay at that waypoint for the specified dwell time. Dwell time may be zero time units if the waypoint is merely a direction and/or speed change point.
3. When the dwell time is up, depart the waypoint and move in the direction of the next waypoint at the desired departure speed. If the desired departure speed cannot be achieved because the platform has suffered degradation in mobility, then move toward the next waypoint at the maximum speed of which the platform is capable.

*Capability definitions for all platforms are documented in appendix D.

†Event types and definitions are documented in appendix G.

‡A script that drives the simulation as it executes the vignette.

4. If at any point during the move between waypoints the platform has a state change (combat damage, component failure, or repair) that changes its speed capability, then its speed is reset to the maximum of which it is then capable but never faster than the desired departure speed from its last node (i.e., there is no logic to make up for lost time). The possibility of speed change includes total immobilization in which case the platform simply comes to a complete stop along its path (and stays there unless a repair later restores some mobility capability).
5. On reaching its next waypoint, the platform resumes this sequence of steps with no. 2. This is repeated until the end of the vignette.

Now for some exceptions:

1. Opposing force (OPFOR) units are also movers, and each has a path of waypoints similar to an OWNFOR platform, but the waypoints represent the unit center rather than a single vehicle (platform) center. In the vignette some of the OPFOR units are supposed to move to an alternative position when they receive fire. This is handled the same way as the moves described previously except that the departure from the current position (one of the waypoints) is triggered by receiving fire, not by expiration of the prescheduled dwell time. When fire is received, the unit's survivors from the incoming fire will depart toward the next waypoint.
2. If a Non-Line-of-Sight Cannon (NLOS-C) conducts a fire mission from a waypoint, it will then depart that waypoint toward its next waypoint at the conclusion of the fire mission (provided that it is not already at its last waypoint). This will be scheduled regardless of whether it has spent the specified dwell time at the waypoint from which it conducted the fire mission. This is to represent a "survivability mini-move" intended to reduce susceptibility to counter-battery fire.
3. If an unmanned aerial vehicle (UAV) is nearing the end of its scheduled flight, and its designated ground recovery platform has lost its capability to recover UAVs, then the UAV's next waypoint will be set to the position of its backup recovery ground platform. If no backup has been specified or the backup has also lost its UAV recovery capability, then the UAV will crash at the end of its scheduled flight time.
4. Unlike a ground platform, if a UAV suffers a total immobilization while in the air, it will crash. It will have no possibility of being repaired and resuming its movement toward its next waypoint in that case.

These characteristics of platform and unit movements are implemented via variables and methods (functions) in the Path and Mover classes (two C++ classes implemented in the SBM).

E.2 Sensors

Each platform may have had one or more sensors on board. The sensor model was elementary and based on a P-infinity curve for each sensor type. For each sensor type, there was an equation of the form

$$P(Acq \mid T) = P_{\infty} \times (1 - \exp(-T/\lambda)), \quad \text{(E-1)}$$

which means that the (cumulative) probability that the sensor acquires the target—given it has been looking at the area in which the target is located for a time of duration T—is given by the expression on the right-hand side. That expression includes two parameters, P_{∞} and λ. The model user could set these parameters to account for sensor type, sensor-to-target range, and background contrast. The first parameter limits the probability of acquisition to a maximum value no matter how long the sensor stares at the target. Making this parameter decrease as a function of range was a simple way to account for the combined effects of poorer sensor performance at increased range, lower probability of having line of sight at increased range,* and atmospheric extinction effects. A more sophisticated model would show each factor in greater detail. The second parameter, λ, is simply the mean time to acquire for those targets whose locations are eventually determined. In the SBM, it too was made a function of range (and sensor type) to give some accounting of the longer time it takes to acquire a target at longer range.

In addition to the acquire functions, each sensor also had an area of coverage, which was described by a maximum sensor range and an angular coverage. The result was a circular wedge (piece of pie)-shaped region of coverage (except in the case where the angular coverage is 360° in which case the wedge simply becomes a full disk). OPFOR units whose centers were outside the coverage of a sensor were not eligible for acquisition by it.

Finally, some sensors were daytime only, whereas others could be used in both day and night conditions.

Determining the probability of acquisition was a fairly complex process for this model. A separate sequence of acquisition check events was generated for each sensor-unit pairing. For each pairing, at initialization of the vignette, a crossing time was calculated, which is when the unit would cross into the area covered by the sensor. This calculation took into account the relative positions and motion of the sensor and unit center as well as the shape of the sensor's coverage region. Whenever a unit entered the sensor's area of coverage, an acquisition process would begin; whenever it left that area (if it ever did), the process was terminated (to be restarted again if the unit again entered that sensor's field of view [FOV]). If the unit was in the sensor's

*This was one of the implicit effects of terrain along with the choice of waypoints and speeds between them. This model represented terrain effects statistically; it did not use a digitized representation of terrain to determine, for example, whether line of sight existed.

FOV at the vignette initialization, then the process began immediately. Otherwise, a future acquisition check was scheduled for the time at which the unit center entered the sensor's FOV. A change in relative velocity of the sensor-unit pair could force a rescheduling (basically canceling an old event and scheduling new one) of the first acquisition check.

At any acquisition check for a given sensor-unit pair, four possibilities could occur with respect to the FOV: the unit may have transitioned from being outside the FOV to being inside it, the unit may have been outside the FOV and remained outside the FOV, the unit may have been inside the FOV and remained inside it, and the unit may have been inside the FOV and transitioned to being outside of it.

E.2.1 From Outside FOV to Inside FOV

If the unit moved into the FOV for the first time* (or re-entered it after moving out), then a cumulative probability of acquisition by the current sensor was initialized (to a value of zero), a uniform random number (between 0 and 1) was drawn and tagged to the current sensor-unit pair, and a future acquisition check was scheduled according to an acquisition time-step value that was input.

If the platform on which the sensor was mounted had multiple sensors on board, then there was an attempt to account for the dependence on what the other sensors acquired. This was accomplished by establishing dependence among the random numbers drawn. The number drawn for the first sensor† on the platform was an independent draw, but for each subsequent sensor on that same platform, the random number drawn was weighted with the random number drawn for the first platform.‡ So the random number stored for the subsequent sensor-unit pair was actually the weighted average of the one drawn for the first sensor and the number drawn when the unit entered the later sensor's FOV. The model user could make the two sensors operate independently by setting the weighting value to zero. If the weighting value were set to one, the sensor of the two having a lower probability of acquisition would acquire the unit only if the one having the higher probability of acquisition already had.§ When a weighting value was assigned between zero and one, variation in the degree of dependence among the sensors** was possible.

E.2.2 From Inside FOV to Inside FOV

When it was time for the next acquisition check for a given sensor-unit pairing and the unit was already inside the sensor's FOV, the model simply (1) updated the cumulative probability that

---

*This includes the possibility of being inside it at the initialization of the vignette.

†The first one according to the order in which they are read. In all of the SBM runs for the demonstration, this was the longest range sensor.

‡Using a weighting value input by the user as part of the sensor data.

§So that if they had the same probability of acquisition parameters and FOV, they would both acquire the target at the exact same time.

**This did not prove especially important in the simple SBM demonstration vignette.

the unit had been acquired by the sensor (using equation E-1 with parameters appropriate for the current range and time increment since last update), (2) compared the results to the random number stored for that sensor-unit pair, and (3) scheduled the next acquisition check according to the input acquisition time step.

If the cumulative probability of acquisition for that sensor-unit pair exceeded the random number, then an acquisition had occurred. In that case, an acquisition report message was sent to the C2V* and its backup. The message gave the details of the perceived unit characteristics (radius, composition, location coordinates, estimated velocity vector, etc.).

E.2.3 From Inside FOV to Outside FOV

If the unit moved from inside to outside the sensor's FOV, then that sensor (but not necessarily that platform if it has other sensors) was considered to have lost the acquisition (and potential target). The cumulative probability of acquisition was reset to zero for this sensor-unit pair and would start from zero should the unit re-enter the FOV at some time in the future. A message was also sent to the C2V and its backup that the sensor has lost the unit. The next acquisition check was scheduled according to the input time step.

E.2.4 From Outside FOV to Outside FOV

If the unit was outside the sensor's FOV before the current acquisition check and remained outside of it, then the model scheduled a future acquisition check according to the input time step.†

As a final note on the subject of sensors, each sensor could suffer capability degradations (or restorations in the case of repairs). When a sensor FOV was reduced (or enlarged) by a degradation (or repair), any resulting change in whether a unit was covered by the sensor was treated in the same manner as though a unit move caused the change.

E.3 Communicators

Each OWNFOR platform was also a communications node that could send and receive messages. The particular types of messages and the means available to send them depended on the type of platform and its role in the OWNFOR mission. The SBM played both data and voice communications. Manned platforms such as the NLOS-C and the command and control vehicle (C2V) (and its backup) were given both data and voice communications; unmanned platforms such as UAVs and armored robotic vehicles (ARVs) were given only data communications.

*If the platform on which the sensor is mounted is the C2V itself, the model still goes through the formality of sending a message to the C2V but with zero delay time and probability of 1.0 of getting through.

†Originally the designer thought this would seldom happen since the first acquisition check (other than at initialization) is not made until the time the unit is expected to cross the FOV boundary. However, with UAVs in the vignette, a unit may come into the UAVs FOV, be reported, then lost and reported as lost, then reacquired by the same UAV later.

There were 14 message types modeled in the SBM:

Message Type 0: "Friendly Force Situation Awareness."

Each OWNFOR platform sent a situation awareness (SA) message to the C2V (with a copy to the C2V's backup) at the end of every SA update interval or whenever it had moved more than the specified SA update distance, whichever came first. Both the update interval and the distance were user inputs. In the case of the runs executed for the demonstration, the SA update interval was 300 s (5 min) and the SA update distance was 500 m.

Each time a platform sent an SA message, it scheduled an event to send the next message at one SA update time interval later; however, if it currently had a nonzero velocity, it also estimated how long it would take to travel the SA update distance. If the latter time was shorter than the regular SA update time interval, then the model would schedule the event to send the next SA update message at that time. As long as a platform's communications link to the C2V remained functional, the C2V's information on the platform should not be older than 300 s, nor its perception of the location off by more than 500 m.*

In the SBM, an SA update message consisted of a time stamp, the platform ID number, its current x-y coordinates, and its current x-y velocity.

Message Type 1: "Common Operating Picture (COP) Update."

Both the C2V and its backup maintained a COP† consisting of the most recent SA information from each platform (as well as the most current information on the OPFOR). The relevant information from this COP was sent to each platform in the OWNFOR to keep it informed of the larger picture.

If the C2V was operational (had data communications), then it sent the COP update. If the C2V had lost data communications capability, then its backup would send the COP update. If both the C2V and its backup had lost data communications, the update was not sent.

In the SBM, the COP update was sent every 300 s (same as the SA update time interval) but at 30 s past the regular SA update time to increase the probability that the COP information sent would be based on the most recently received SA update messages.‡

Rather than tailor the COP message to what each OWNFOR platform needs to know, the SBM just sent the entire COP to every platform and then used what was necessary based on the most recent COP message that it had received. The information in the COP consisted not only of the

*No more than 500 m plus any error in self-locating by the sending platform.

†The COP is sometimes called "common relevant operating picture" or "CROP" in older literature.

‡The SA update messages triggered by movements are not synchronized with the regular update intervals, so some of the information in the COP could be older.

most recent SA message received from each platform, but also of the most recent acquisition information on the location of OPFOR units, and the most recent reports of platform capability status.

Message Type 2: "Capability Status Update."

Each time a platform had a change in state because of a component failure, damage, or repair, the updated current capabilities were reported in a message to the C2V (copied to its backup).* Having this information allowed the C2V to assign fire missions only to capable NLOS-C and, had the vignette not been deterministically scripted, would have permitted reallocation of sensor assets if, for example, a UAV were shot down.

Message Type 3: "Activity Update."

This was originally intended for the platforms to report their current activities to the C2V, so that the unit could make plan adjustments if some tasks were failing.

Message Type 4: "Acquisition Report."

When the cumulative probability of a sensor acquiring an OPFOR unit exceeded the random number set at the start of the acquisition process for that sensor-unit pairing, then the platform on which the sensor was mounted would send a message to the C2V (with a copy to its backup). This message simply said that an OPFOR unit had been acquired with a center at estimated x-y coordinates.† This message type also reported the loss of acquisition of an OPFOR unit if it moved out of sensor coverage or the sensor was degraded.

Message Type 5: "Call for Fire."

When the C2V (or its backup) received an acquisition report of an OPFOR unit, it would look at the COP to determine which NLOS-C had the target in range, which were not busy with other fire missions (or if they all were busy, which would be available soonest), which had sufficient ammunition, and which had firepower capability. It would then pick two of them to conduct the fire mission.‡ The C2V (or its backup) would then send a "call for fire" to those two NLOS-Cs. The message would consist of the perceived OPFOR unit coordinates and velocity.

---

*In this version of the SBM, it was assumed that the onboard diagnostic equipment for each platform had perfect self-awareness of its own component-level state and corresponding capabilities. This is an area for more realistic model representation in the future.

†Initially in the design, there was some thought to differentiating between perceived OPFOR unit information and ground truth; however, since the logic in the SBM called for a fire mission against every OPFOR unit acquired, passing information like the perceived size and type of unit had no affect on the decision to fire. The only difference between the perceived situation and ground truth that survived into the running SBM was the modeling of target location error discussed in appendix F, Simulation Events, Event type 8, round arrives at aim point.

‡Logic was added to make sure the two selected were from the same platoon. The six NLOS-C were divided into two platoons of three each.

Message Type 6: "Fired, Over."

On receipt of a call for fire message, an NLOS-C would fire a round at the specified coordinates (or, if the OPFOR unit were perceived to be moving, at the predicted location when the round would arrive). There was a slight delay between the receipt of the message and firing the round, but once the round was fired, the NLOS-C sent a "fired, over" message to the C2V (and its backup) to indicate the round was fired.*

Message Type 7: "Request Battle Damage Assessment (BDA)."

In the early design of the model, it was planned that the C2V would potentially follow each fire mission with an order to one or more of the sensors (probably a UAV) to perform BDA.

Message Type 8: "BDA Report."

What the UAV (or other sensor) would send back as it executed a BDA request.

Message Type 9: "Cease Fire."

This message was sent by the C2V (or its backup) to the participating NLOS-C when a fire mission had been completed. A fire mission in the SBM was considered completed when the C2V had received the "fire, over" messages from all of the NLOS-Cs assigned to the fire mission.

Message Type 10: "Target Update."

A sensor would continue to monitor an OPFOR unit in its FOV even after the platform owning the sensor had sent an initial acquisition report on it. This message type was sent to the C2V (and copied to its backup) to indicate the OPFOR unit was still acquired (and to update its perceived location if it had moved).

Message Type 11, "Copy Call for Fire."

When the C2V sent a "call for fire" message, it copied its backup on the fire mission details. Should the backup need to take over control of the OWNFOR, it would have the information to send a cease fire when appropriate and not duplicate a call for fire against the target.

*This is putting the C2V in a fire direction center–like role, which may turn out not to be how the NLOS-C will be controlled under the Future Combat System concept. Also, since smart munitions were used, only one round was fired per NLOS-C per fire mission, so this message also informed the C2V that this NLOS-C had not only finished firing that round, but had also finished its part of the fire mission.

Message Type 12: "Cannot Fire."

On rare occasions an NLOS-C may receive a call for fire and then suffer a failure or combat damage that prevented it from firing the round.* If this happened, the NLOS-C would send the C2V (or its backup) a message that it could not fire the mission in its current state.

Message Type 13: "Fire Mission Done Copy."

This message type from the C2V indicates to the backup that the fire mission is completed. Should the backup need to take over control of the OPFOR, it would not try to continue a fire mission that it thought was still ongoing.

This completes the description of the message types supported in the SBM. Each communications node (that is, each OWNFOR platform) would try to send its message types using data communications (rather than voice) if both sender and receiver had functioning data communications. If data communications were disabled at the sending transmitter or were perceived† by the sender to be disabled at the receiver, then the sending platform would use voice communications provided it had functional voice communications and perceived that the receiver also had functioning voice communications capability.‡ If there was no functional communications mode that was compatible at both the desired sending node and the desired receiving node, then the message could not be sent.§

E.4 Shooters

In addition to platforms that move, acquire, and communicate, the OWNFOR included some platforms that shoot. In the vignette developed for the SBM, only the NLOS-C actually fired their weapons.

Much of the detail of the shooter class is covered on in the discussion of event types (appendix F) and message types in the previous section of this appendix. When the C2V (or its backup if it has control of the OWNFOR) received a target acquisition report of an OPFOR unit from a sensor, it decides whether to engage the target. In the SBM, the decision was always to engage the target unless a fire mission was already in progress as a result of the current acquisition.

The fire missions in the SBM each consist of firing two smart munition rounds against the target unit. Given that the target OPFOR units each consisted of only one or two vehicles (either armored personnel carriers or self-propelled howitzers), this seemed like an economically sound use of ammunition.

---

*Or the C2V may have been operating from an outdated COP when it selected that NLOS-C to participate in the fire mission.

†Perception of another node's communications capability state was based on the latest COP update received, which propagated the latest update of each platform's capability state (including those related to communications).

‡SA and COP update messages (types 0 and 1 in the SBM) were never sent by voice.

§In the case of the UAV and ARV, which are unmanned, there was never any voice communications capability.

The C2V would assign each fire mission to two NLOS-Cs, each instructed to fire one round. In deciding which NLOS-C should fire the mission, the C2V would use its COP to eliminate any NLOS-Cs that were out of ammunition, did not have the target within range, could not communicate, or had a current degradation that prevented them from firing. From among those that remain, the two that were available to fire soonest and belonged to the same platoon were selected.* An event sending a "call for fire" message to each selected NLOS-C would then be scheduled. When executed, each such event would schedule a later event receiving the message at each selected NLOS-C. Receipt of that message would, after appropriate delay, result in a round fired at the appropriate point.† Once fired, a "round arrives" event would be scheduled to occur after the time of flight has elapsed. When the "round arrives" event was executed, a burst point was generated for the round, taking into account the delivery accuracy of the weapon for that range. The bursting round was simulated by dispensing two smart submunitions‡ at points randomly generated in a circular disk of specified radius.

From each of the submunition dispense points, a footprint was projected onto the ground plane below. To model the acquisition and target selection by each submunition, a simple first-order model was used. It used the false target density, the number of dead hulks§ in the footprint, and the probability of acquiring each live target type currently in the footprint to obtain an expected number of things acquired (false target, dead hulk, or live target). The model then used that expected value as the parameter in a Poisson distribution to determine the probability of acquiring nothing at all; if that were the case, an appropriate counter was incremented. If, however, the model acquired something, then the probability that it selected each type of thing (false target, dead hulk, or live target) to attack was set to be proportional to the expected number of each type acquired (scaled to sum to 1.0). A random number was drawn to determine which type selection to attack. If the selection was a false target or a dead hulk, then an appropriate counter was incremented. If the selection was a live target to attack, then a further random number was drawn. This number was compared to the probability of hit and kill, given selection, to determine whether a kill was achieved. If killed, a counter was incremented, and the killed target vehicle was added to the list of dead hulks for future submunitions to acquire. If not killed, then a counter of submunitions that unsuccessfully attacked a real target was incremented.**

*The six NLOS-Cs were assigned to two platoons of three each. So the logic described really chooses two from the platoon that are capable of responding to the call for fire and responding soonest. Of course, it is theoretically possible that all of the NLOS-C could be suffering from firepower capability degradations resulting in no fire mission being assigned at the time.

†That point will be given by fixed coordinates reported by the acquisition sensor if the target OPFOR unit was perceived to be stationary when acquired. If it was perceived to be moving, then its estimated velocity and the time of flight of the round will be used to calculate its location when the round will arrive, and the aim point will be selected accordingly.

‡For a total of four on the target for each fire mission if all goes according to plan.

§From previous fire missions.

**If the OPFOR unit is not already at its final waypoint, then whatever OPFOR platforms survive the fire will be scheduled to depart for their next waypoint immediately, regardless of the dwell time value.

Appendix F. Simulation Events

The Storyboard Model (SBM) was an event-sequenced simulation executing events from a number of scripts and dynamically generating some random events. Two of the scripts were developed separately but then merged into a single script file of time-ordered, externally generated events. One scripted the tasks demanded of each platform as the vignette unfolded, and the other scripted the combat damage events, failure events, and repair events used to drive the physical state changes of the platforms in the own force (OWNFOR) (and ultimately their capabilities). The merged script set the vignette in motion and drove both the varying tasks demanded of the OWNFOR (the left side of the Missions and Means Framework [MMF] diagram) and the varying capabilities of the platforms in the force (the right side of the MMF diagram). There was also a scripted schedule of OWNFOR platform moves and opposing force (OPFOR) unit moves to be executed; the movement schedule could have been included in the script file but was needed earlier in the model development and was included as a part of another file.*

The model also required a number of performance parameters describing the capabilities of undamaged platforms, a low-resolution description of the OPFOR, the communications network, etc.

There were two common methods used to advance time as a simulation executed: time stepping and event sequencing. Each has advantages and disadvantages.

The time-stepped approach divided time into small intervals or steps. At the start of the first interval, all simulation entities were initialized. Then at the start of each subsequent step, the states of all entities were updated based on their states at the start of the previous interval and the activities in which they were engaged (e.g., shooting, moving, communicating) during the interval.

The event-sequenced approach also initialized the entities to their starting states. However, a time-ordered event queue was referenced to determine what event was to be executed next.† The event notice (the information describing the event) was removed from the queue, and the entities it affected had their states (including the activities they were engaged in) modified as required by the event. Each event potentially placed new events on the event queue. For example, an event "entity 17 fires a round at coordinates x,y" would be executed, entity 17 would have its

*Both script files and other files of model parameters are described in appendix H.

†In most cases, the initialization will include placing a few events on the event queue to get things started. In the case of the SBM, the first external event from the script file is placed on the queue as well as several internally generated events scheduled at initialization.

ammunition decremented, a time of flight and impact or burst point for the round would be generated, and a "round arrives" event would be placed on the event queue with the appropriate future time stamp and relevant parameters (e.g., impact/burst point coordinates, round type). As events are taken from the event queue and executed, eventually that "round arrives" event would be executed. At that time, the model would determine whether there were any entities in the area of the impact or burst point that would be damaged by the round as well as call a function to calculate such damage, taking into account the parameters (e.g., round type) stored in the event notice and other relevant variables (target type, posture, location).

The event-sequenced approach also required that there be a way to remove events that need to be cancelled. If in the previous example an active protection system were to intercept the round before it reached its burst/impact point, then the "round arrives" event would need to be cancelled and removed from the event queue before it was executed.

The advantage of the event-sequenced method was that if the events are far enough apart in time, one does not waste a lot of time doing complete updates of all the entities' states when very little was changing. This was often the case when the number of entities was small. When the number of entities was large, there were so many events being scheduled (and removed) that time stepping was just as efficient.

Time stepping could be more easily parallelized at the top level than event sequencing, so both approaches survive in the simulation community. Of course, it is possible to use a hybrid of the two approaches where some things are updated at regular intervals (time stepping) and others are updated at randomly distributed or otherwise irregular inter-event times.

For the most part, the SBM was an event-sequenced simulation. Time was advanced from event to event, and the state of the systems affected by each event was adjusted accordingly as it was executed. There was a time-stepped nature to some of the events, such as the acquisition process where updates of the probability that a sensor had acquired a target occur at regular intervals once the target has entered the sensor's field of view (FOV) (by executing an event). Some events came from the external scripts (covered in appendix G), and others are scheduled on the fly in reaction to the state changes that occurred as the scripts unfolded.

A replication of the SBM vignette would be played in the following manner:

1. The OWNFOR platforms and the OPFOR units are placed in their initial positions and (if appropriate) set in motion toward their next waypoints at the specified speed. For each moving platform, an arrival time event is scheduled according to its speed and the distance to its next waypoint. For each stationary platform, a departure event is scheduled for it to leave its current position (start point), setting the vignette in motion.
2. As the platforms and units move about the vignette area, they execute events in time order. External events, such as change in platform state (components lost to combat damage, components lost to failure, and components repaired), are represented by changing the

current capability state* of the platform. External events that represent the platform task load include the requirement to start a new task or the ending of a current task. So as time progresses, each platform has a varying task load and a varying set of residual capabilities (representing respectively the left and right sides of the MMF diagram in figure A-1). In addition, the SBM is executing routine activities, such as moving (as best it can), acquiring, sending messages, and responding to calls for fire.

Because of the way the model was designed and the vignette was developed, there are some deviations from what one would like in such a model. For example, there was no way for an externally developed vignette script to account for exactly when a platform will acquire a target while still allowing for a fully dynamic acquisition process within the model.†

The model was designed to handle the following 18 event types:

Event Type 0: Platform departs a path node (or waypoint).

The simulation was initialized with each OWNFOR platform and OPFOR unit having a starting position. For each platform, a "depart" event was scheduled and placed on the event queue at the time the departure was to occur. Those entities that began moving at the very start of the simulation had their departures scheduled zero time units from the initialization.

When a departure event was executed, the entity (OWNFOR platform or OPFOR unit) was assigned a velocity. The speed was the desired departure speed read in for that node unless the entity was no longer capable of that speed because of degradations. In the latter case, the speed was the maximum capable. The direction of the velocity was always toward the next node‡ on the path read in for that entity. An arrival time at the next node was calculated, and an "arrival" event was scheduled to occur at that time. If there were further degradation or repair to the entity, its speed might change before it reached the next node. In this case, the arrival event was cancelled, a new arrival time at the node was calculated based on the new speed, and an arrival event was put on the event queue at that time.§

Event Type 1: Platform arrives at a path node (of waypoint).

When an OWNFOR platform or OPFOR unit reached a path node (waypoint), an arrival event was executed. This event scheduled a "depart" event from that node to occur after a "dwell time" at that node. The dwell time at a node might be zero, if the node were simply there as a

*As represented by the elements of capability degradation (ECDs) in effect for that platform. (ECDs are defined in appendix D).

†This can result in the script requiring a platform to have surveillance capability for a 1-h interval even though in a particular replication of the vignette, it acquires all OPFOR units in the area in the first 15 min of the interval.

‡Nodes are also called waypoints.

§If the entity is completely immobilized, an arrival event is scheduled for a time beyond the end of the vignette.

point at which direction or speed changed. However, the vignette developer may supply a nonzero dwell time if desired.*

Event Type 2: Platform changes speed.

If an OWNFOR platform suffered a damage or failure event or underwent a repair event that changed its maximum forward speed, a "speed change" event was then scheduled to occur immediately (or at a randomly generated future time† if the degradation was a "stop after time T"). The speed change event simply reset the entity's speed toward the next node to be either the desired speed read in for that path segment or the maximum possible forward speed in its new state, whichever was less.‡ The direction was still toward the next originally scheduled node.

Event Type 3: Acquisition check.

For each OPFOR unit and OWNFOR platform with sensors, an "acquisition check" event was scheduled for each sensor on the OWNFOR platform.

If the OPFOR unit was initialized in the FOV of an OWNFOR sensor at the start of the vignette, then the acquisition process began scheduling "acquisition checks" immediately. For those sensors for which the OPFOR unit was outside the FOV at the start of the vignette, a time was calculated for the OPFOR unit to enter the FOV based on their projected trajectories. Since both the OPFOR unit and the OWNFOR platform with sensor(s) may be moving, this time calculation considered both the velocity at which each platform was moving and the size and shape of the sensor's FOV. Since entire sensor FOVs were either circles or circular wedges (pieces of pie), the OPFOR unit could cross the boundary of the FOV a maximum of four times. An "acquisition check" event was scheduled for the earliest of those boundary crossing times not yet in the past (and assuming that there was one). Of course, if either the OPFOR unit or the OWNFOR platform with the sensor changes velocity before the boundary crossing, the "acquisition check" event would be cancelled and, if appropriate, another one scheduled for a different time based on the new relative velocity.

There were two cases of the "acquisition check" event that need to be distinguished. One case is when the OPFOR unit first became subject to acquisition by the sensor (either by being initialized in its FOV or by entering the FOV as the simulation progresses), and the other was when the acquisition process was being updated.

---

*For example, to represent a platform in an overwatch position.

†Future time was randomly generated from a truncated exponential distribution with a minimum of T and maximum of 5T. Later in the MMF project, the "stop after time T" degradation was replaced with a "stop within time T" degradation; however, that occurred after the first generation of the SBM described in this report.

‡If the platform had been traveling at less than the desired speed and was restored to a speed capability in excess of the desired speed for that path segment, there was no logic to speed up beyond the desired speed to make up for lost time. Instead the repaired platform simply traveled at the desired speed from that time until another event changes its speed.

When a unit first became subject to acquisition by a sensor,* a random number between 0 and 1 was drawn and tagged to that sensor-unit pair. If it was the first sensor on a platform to have the unit in its FOV, then the random number was unmodified. If another sensor on the same platform already had the unit in its FOV, then a random number was still drawn, but it was weighted with the random number drawn for the first sensor to account for the dependence among sensors. The degree of weighting was a user input (details are described in appendix G). Whether it was the first sensor or a later one to have the unit in its FOV, another "acquisition check" was scheduled at one acquisition interval (an input time interval) later.

If the "acquisition check" event was not the initial entry of that unit into the sensor FOV, then it did not draw a random number. Instead, the cumulative probability of being acquired was updated and compared to the previously set random number for that sensor-unit pair. If the cumulative probability was greater than the random number, the unit was acquired, and an acquisition report message was scheduled to be sent.† If the cumulative probability was still less than the random number, no such message was sent. In both cases, a subsequent acquisition check event was scheduled one acquisition interval later.

If the "acquisition check" determined that the unit was still in the sensor FOV but had moved or otherwise changed state, an "acquisition update" message was scheduled to be sent. If the "acquisition check" determined that the unit had moved out of the FOV since the last check or the sensor had lost capability, then an "acquisition update" message was scheduled to be sent, indicating that the unit had been "lost." This last situation rarely occurred in the SBM vignette except in the case of unmanned aerial vehicles (UAVs) whose FOVs were constantly moving.

Event Type 4: Platform sends message.

When circumstances were right, each OWNFOR platform scheduled a message to be sent. For many message types, this was scheduled immediately.‡ However, for situation awareness (SA) updates, sending the next such message was scheduled each time one was sent.

Event Type 5: Platform receives message.

When a message was sent, its receipt was scheduled (assuming sender and receiver had compatible communications capability at that time). A random delay time based on message type and communications type (data or voice) was sampled, and the receipt of the message was scheduled at that time.

---

*Or again enters the sensor's FOV after previously leaving it.

†There was a modeling assumption here that even if the platform on which the sensor was mounted has no human crew (was robotic), that it had a good enough automatic target recognition system to make an acquisition. An alternative assumption would be that the acquisition was made by a human operator (on the command and control vehicle [C2V]) who was monitoring the sensor feed. In the latter case, a more detailed representation of communications bandwidth between sensor platform and operator should be added for greater realism.

‡The various types of messages are detailed in appendix E, section E-1.

Receipt of the message caused each OWNFOR platform to update its perception of the operation. In some cases, such as messages reporting the acquisition of an OPFOR unit, the message might trigger a further event, such as sending a "call for fire" message.

Event Type 6: Platform acquires something.

The functionality originally intended for event type 6 was incorporated into event type 3 as the model was developed. In those cases (of event type 3) where the cumulative probability of acquisition became larger than the random number for the given sensor-target pairing, an acquisition occurred and a message reporting it was scheduled. Event type 6 was, therefore, unnecessary.

Event Type 7: Platform fires a round.

This event was scheduled when a Non-Line-of-Sight Cannon (NLOS-C) received a "call for fire" message from the C2V (or its backup).* A check was done to make sure the NLOS-C can actually engage the target, then a delay time was drawn, and the fire round event was scheduled.

When this event was executed, the round was fired, and a time of flight was estimated to the target. If the target was perceived to be moving, then the aim point was selected to match the point where the target was expected to be when the round arrived. This involved solving a system of equations giving the position of the target (as estimated) and the position of the round as a function of time. The resulting solution time and the x,y coordinates of the intersection point gave the time of flight and the aim point. A target location error model was included to account for the fact that the target may not move as predicted.[1]

Event Type 8: Round arrives at aim point.

When the NLOS-C fired a round, a "round arrives" event was scheduled.

When the round arrival event was executed, the round's burst point was set (it is the aim point plus a vector of delivery errors) and the submunitions were dispensed. Each submunition was placed according to a dispense pattern distribution (uniform random within a circular disk of specified radius at present), and its search pattern (or "footprint") was projected onto the ground plane.

*There was logic in the model so that the C2V (or its backup) would send the fire mission to those NLOS-Cs available soonest (not busy with other fire missions), in range, having ammunition, and perceived as functional (not suffering any degradation that would prevent firing the mission). Only if the NLOS-C selected to fire had suffered degradation between receiving the call for fire and actually firing would there be a need to cancel this event type.

[1]The error model used the algorithm developed by J. Chernick (*Moving Target Location Errors for Ground Vehicles*; Technical Report TR-309; U.S. Army Materiel Systems Analysis Activity: Aberdeen Proving Ground, MD, August 1980).

An expected number of false targets per unit area were used to generate a Poisson-distributed random number of false targets in the footprint. A linked list of dead hulks from past kills in the vignette was checked to determine how many of those were in the submunitions footprint. The live OPFOR platforms were then checked. A total "expected number of things acquired" was generated by weighting the number of items in each category (false targets, dead hulks, live platforms) by its probability of being acquired. Then a random number was drawn to determine which, if any, of the items were attacked by the submunition. If the submunition attacked a false target, a dead hulk, or nothing at all, then that event was tallied for the output statistics. If it attacked a live target, then a probability of kill given attack was compared to a random number to determine whether a kill was achieved. If it was, the live target was converted to a dead hulk (and, as such, could attract future submunitions); if it was not, the submunition-target interaction was scored as a "no kill" for output statistics, and the target continued to be treated as "live" for any subsequent submunitions that may encounter it.

In some cases, the "round arrives" event may trigger a "depart" event by the unit (if any) in the area. This was accomplished by designating specific OPFOR units to behave in that way and determine where they would move to next. Of course, only those platforms within the OPFOR unit that were still functional would make such a move; dead hulks remained where they were killed.

Event Type 9: Sensor orientation change.

When a sensor acquired a target, it reoriented its FOV so that the centerline of the wedge was aligned on the most recently acquired target. This behavior really ought to be improved to something more realistic, but it proved to be almost irrelevant because no targets were lost as a result of sensor reorientations.

Event Type 10: External event.

External events were read from a script and inserted in the event queue. The event could be combat damage to some components, failure of a component, repair of a component, or a change (starting or ending a task) in the set of tasks the platform was supposed to be executing. In addition to these external events representing things that happen to the force in the vignette, there were two external events that were model control features; one was an event to initialize all of the OWNFOR platforms in pristine condition at the beginning of the vignette and the other was a signal to the program that the vignette had ended.

Each time an external event was next on the event queue, it was executed just like an internally generated event. Then the next external event was read from the script and inserted in the event queue so that it could be executed at the appropriate time.

Event Type 11: Command and control of OWNFOR transfer.

This event was scheduled to occur X minutes after the C2V loses data communications or Y minutes after it lost both data and voice communications (where Y > X). The assumption was that if it lost data only, it would use voice to notify its backup (if any) within X minutes to take over. If it lost both digital and voice, the backup would figure this out within Y minutes and take over anyway.

Both the C2V and its backup were copied on all SA messages and target acquisitions; however, only the one currently having command and control sent calls for fire.

If the C2V lost data communications but that capability was later restored, it would take over control again from its backup.

Event Type 12: Change from night to day (or vice versa).

At certain times scripted into the vignette (as an external event), there might be a change from night to day or from day to night. This event was executed by simply toggling a global flag indicating whether it was currently day or night. The effect on the simulation was that some sensors might have different ranges (including possibly zero range) depending on the light conditions. There was no more sophistication in this model than just switching between the two states.*

Event Type 13: Print move report.

This event caused a report of the current location and velocity of each OWNFOR platform and each OPFOR unit (center) to be printed to a separate file. It then scheduled the next such printout in 900 s (15 min) of simulation time, which resulted in snapshots of the mover locations every 15 min. The file was used early in model development for debugging the movement schedule and not used thereafter.

Event Type 14: Launch of a UAV.

Prior to UAV launching, its path and coordinates match those of the platform transporting it.† When launched, it began its own independent path. The original intent was that the UAV launch would only occur if the platform carrying it still had the capability to launch. However, as the capability to launch was not one of the capabilities represented in the model, the launch only failed if the UAV had lost mobility (ability to fly) because of damage or failure prior to launch.

*There was no gradual change in sensor capability as light conditions change during dawn or dusk.

†For those UAVs already launched before the vignette begins, there is no launch event. They are treated as independent platforms until recovered.

Event Type 15: Recovery of a UAV.

At certain points in the vignette, there were externally scheduled UAV recoveries. In this event, the UAV had returned to its controlling ground platform, and if UAV recovery capability was intact, the recovery event was executed.* There is also a dynamically scheduled UAV recovery: if a UAV sensed it was about to crash (has a stop after time T event), it would try to reach its controlling ground platform before time T was up.†

Event Type 16: Risk table change (change in phase of battle).

This was a scripted event that modified the set of events considered critical during various phases of the vignette. Its primary function was changing the ratings for which task failures were considered acceptable risks (i.e., which tasks were critical) and thus how the green-yellow-red ratings were changed as a function of the battle phase (see appendix H, section H.3).

Event Type 17: Update statistics.

When this event was executed, the program updated the tallies that would later be used to calculate the statistics described in appendix H, section H.5. The event then scheduled the next event of the same type in 60 s so that the data collected would be updated every minute throughout the simulation.

*UAV recovery was not a separately represented capability in the model. The capability to recover was only lost if the controlling ground platform had suffered a catastrophic kill.

†The UAV was considered to have enough onboard diagnostic capability to determine whether it needed to return to its controller quickly or not. This was a debatable assumption to be sure; however, it occurred so infrequently that it had little (if any) effect in the actual cases run for the demonstration.

INTENTIONALLY LEFT BLANK.

Appendix G. Model Inputs

The Storyboard Model (SBM) required multiple files as input. This appendix presents a description of the files and how they were generated, reformatted, and merged in preparation for use by the SBM. Four files fed directly into the SBM. The largest file was generated by merging two separately developed time-ordered event files that constituted most of the model "script."

G.1 Time-Ordered Event List: Level 4

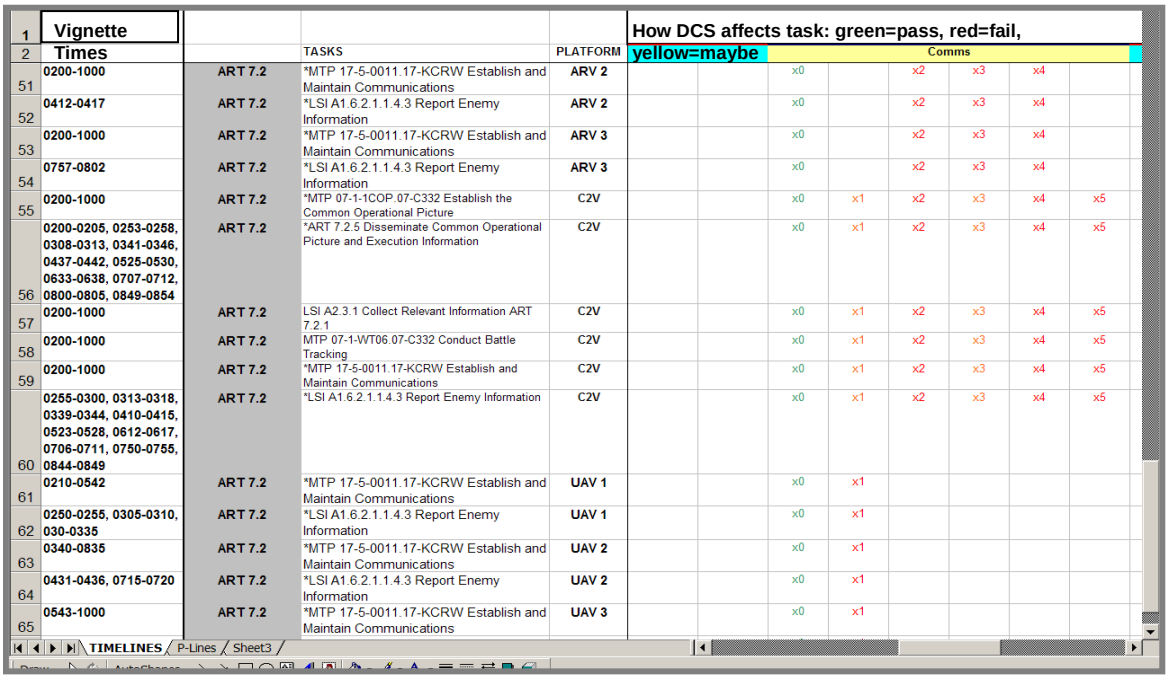
The generic description time-ordered event list (TOEL) actually applied to a number of the files used in preparing the script for the SBM. In this appendix, TOEL will be used as the name for the vignette description.* The TOEL outlined the major events in the vignette (see figure G-1 for an excerpt).

SEQ #	TIME	ACTIVITY
	0200-0400	PHASE I
P101		MCS A in AP Muldraugh and preparing for movement to OBJ APPLE
P102		C2V establishes ACA MAPLE, min alt 500 ft AGL, max alt 1000 ft AGL, ES860930, ET850050, ET880050, ES890940, eff 0200-0600
P103		C2V disseminates ACA MAPLE coordinates to CAB
P104		C2V launches UAV 1 from AP Muldraugh vic ES864943
P105		UAV 1 travels from AP Muldraugh (ES 864943) to perform route reconnaissance. Route ACPs: ES865945 (SP) to ET 883011 to ES866957 to ET875045 (OBJ APPLE)
P106		C2V monitors incoming data from UAV 1 visual and sensor feeds as it travels north along AXIS MAPLE
P107		MCS A plts conduct perimeter security in AP Muldraugh with their respective ARV-Rs
P108		MCS A plts perform precombat checks in preparation for movement north to OBJ APPLE
P109		UAV 1 remains on OBJ APPLE and conducts reconnaissance of TAI's 1 and 2. UAV 1 Route ACPs: ET876050, ET856040, ET880005, ET876050. UAV performs continuous loop on OBJ APPLE.
P110		C2V monitors incoming data from UAV 1 visual and sensors feed as it conducts reconnaissance of OBJ APPLE
P111	Interaction 1	*UAV 1 detects suspected enemy activity vic TAI 2 with IR sensor*
P112		UAV 1 sends sensor report to C2V
P113		C2V receives IR sensor report of enemy activity vic TAI 2
P114		C2V updates the COP and informs MCS A Cdr
P115		Updated COP disseminated to higher and lower echelons
P116		C2V continues to monitor UAV 1 sensor feeds
P117		C2V tasks UAV 1 to stare at suspected enemy activity position to achieve better fidelity for target identification
P118		UAV 1 IFF sensor does not confirm friendly force
P119	Interaction 2	*UAV 1 detects elements of a suspected enemy INF squad vic ET 877036*
P120		UAV 1 transmits information to C2V
P121		C2V receives UAV 1 information and cannot confirm or deny enemy forces and continues to monitor activity
P122		UAV 1 maintains surveillance of TAI's 1 and 2 and OBJ APPLE
P123		MCS plts begin to assemble in order of march formation and prepare for tactical movement
P124		NLOS-C/M receives updated COP and plans targeting data for TAI 2.
P125		C2V and MCS A Hq prepares for movement toward OBJ APPLE
P126		MCS A plts task ARV-R 2 & 3 to move north along AXIS MAPLE with a limit of advance of 3km from plt main body and conduct reconnaissance. ARV-R 2 will travel route ES871948 (SP), ES873966, ES876987, ET875008, ET878018 ARV-R 3 will travel route ES862951 (

Figure G-1. Excerpt of time-ordered event list (TOEL).

*The TOEL was developed by Dynamics Research Corporation (DRC)—subject matter experts in military operations.

The TOEL, however, described the vignette at a higher level than the SBM could use directly. The TOEL had to be translated into a time-ordered list of tasks assigned to each own force (OWNFOR) platform in the vignette. To that end, another spreadsheet was developed (as shown in figure G-2) with a row for each task type and platform pairing.*



	Vignette				How DCS affects task: green=pass, red=fail,						
2	Times		TASKS	PLATFORM	yellow=maybe		Comms				
51	0200-1000	ART 7.2	*MTP 17-5-0011.17-KCRW Establish and Maintain Communications	ARV 2			x0		x2	x3	x4	
52	0412-0417	ART 7.2	*LSI A1.6.2.1.1.4.3 Report Enemy Information	ARV 2			x0		x2	x3	x4	
53	0200-1000	ART 7.2	*MTP 17-5-0011.17-KCRW Establish and Maintain Communications	ARV 3			x0		x2	x3	x4	
54	0757-0802	ART 7.2	*LSI A1.6.2.1.1.4.3 Report Enemy Information	ARV 3			x0		x2	x3	x4	
55	0200-1000	ART 7.2	*MTP 07-1-1COP.07-C332 Establish the Common Operational Picture	C2V			x0	x1	x2	x3	x4	x5
56	0200-0205, 0253-0258, 0308-0313, 0341-0346, 0437-0442, 0525-0530, 0633-0638, 0707-0712, 0800-0805, 0849-0854	ART 7.2	*ART 7.2.5 Disseminate Common Operational Picture and Execution Information	C2V			x0	x1	x2	x3	x4	x5
57	0200-1000	ART 7.2	LSI A2.3.1 Collect Relevant Information ART 7.2.1	C2V			x0	x1	x2	x3	x4	x5
58	0200-1000	ART 7.2	MTP 07-1-WT06.07-C332 Conduct Battle Tracking	C2V			x0	x1	x2	x3	x4	x5
59	0200-1000	ART 7.2	*MTP 17-5-0011.17-KCRW Establish and Maintain Communications	C2V			x0	x1	x2	x3	x4	x5
60	0255-0300, 0313-0318, 0339-0344, 0410-0415, 0523-0528, 0612-0617, 0706-0711, 0750-0755, 0844-0849	ART 7.2	*LSI A1.6.2.1.1.4.3 Report Enemy Information	C2V			x0	x1	x2	x3	x4	x5
61	0210-0542	ART 7.2	*MTP 17-5-0011.17-KCRW Establish and Maintain Communications	UAV 1			x0	x1				
62	0250-0255, 0305-0310, 030-0335	ART 7.2	*LSI A1.6.2.1.1.4.3 Report Enemy Information	UAV 1			x0	x1				
63	0340-0835	ART 7.2	*MTP 17-5-0011.17-KCRW Establish and Maintain Communications	UAV 2			x0	x1				
64	0431-0436, 0715-0720	ART 7.2	*LSI A1.6.2.1.1.4.3 Report Enemy Information	UAV 2			x0	x1				
65	0543-1000	ART 7.2	*MTP 17-5-0011.17-KCRW Establish and Maintain Communications	UAV 3			x0	x1				

TIMELINES | P-Lines | Sheet3

Figure G-2. Task requirements to platform capability mapping.

In the spreadsheet, the TOEL had been converted to a set of tasks applicable to each platform in the vignette, and a time interval (or set of intervals) had been specified (first column) during which the platform must be capable of performing each task. In addition, the platform-functional attribute requirements† were added into the spreadsheet (the columns in this example with x0, x1, x2, x3, and x5 in some cells). The spreadsheet thus contained the information on which tasks each platform must be able to perform during each time interval as well the degradations that would cause it to fail the task.‡ The next steps put this information into a format that SBM can use.

*In some cases it was a task- and platform-type pairing with all platforms of a given type assigned the same task.

†The elements of capability degradation (ECD) were developed by Survivability/Lethality Analysis Directorate engineers.

‡More specifically, the red ECDs on the spreadsheet were ones that, if triggered, would cause task failure. In some instances (none in this excerpt), there may have been combinations that would trigger task failure. The orange ECDs were those that, if lost, would make accomplishment of the task risky but not impossible.

First the spreadsheet was converted into a semicolon delimited text file. Then an AWK script was run to generate a line for each start time and another line for each stop time indicating the platform, platform type, task number (as numbered 0 through N for the model), and some alphanumeric strings with the platform designator and task description. The file was sorted so that the lines would occur in time sequence instead of grouped by task type as in the spreadsheet. Finally, the task description alphanumeric strings were modified so that any internal blanks were replaced by underscores (this makes C++ string handling simpler). In addition, the time stamps were converted from the common military format to a time in seconds.* An excerpt of the resulting file is in figure G-3.

```
 9480.0000   4   3   2  12  UAV1  Conduct_Surveillance_ART1.3.4

 9480.0000   4   4   2  12  UAV2  Conduct_Surveillance_ART1.3.4

 9600.0000   5   3   2  10  UAV1  Conduct_Tactical_Reconnaissance_ART1.3.3

 9960.0000   4   3   2  14  UAV1  Detect_and_Locate_Surface_Targets_ART3.2

10200.0000   4   3   2   6  UAV1  Report_Enemy_Information

10260.0000   5   3   2  14  UAV1  Detect_and_Locate_Surface_Targets_ART3.2

10380.0000   4   6   3   1  C2V   Disseminate_Common_Operational_Picture_

and_Execution_Information
```

Figure G-3. Excerpt from task start-stop file.

To improve the readability of this figure, a blank row has been inserted following each line that had to be wrapped; the actual file read by the SBM did not have the blank lines. Each line began with a time in seconds indicating the time at which the task in question started or stopped. The second field had a "4" if it was the start of a task; it had a "5" if it indicated the end of a task.† The third field indicated the platform number (as assigned in the vignette) of which the task was required; the fourth field indicated the platform type. The fifth field indicated the task number (as assigned in the SBM), the sixth field was merely an alphanumeric identifier for the platform, and the seventh field is an alphanumeric string describing the task (in the language of the Army Universal Task List [AUTL], Universal Joint Task List [UJTL], or Single Integrated Task List [SITL]).

*To avoid having to do frequent time unit conversions in the model, all internal times were in seconds. For example, a time of 0200 in the vignette becomes 7200.00 s past midnight on the simulation clock.

†To be more precise, these were the start and end times of the interval during which the platform may be required to perform the task if the vignette simulation unfolded as planned in the TOEL. Actually, execution of the vignette could sometimes differ from the plan. This was a limitation of the scripted modeling approach.

The other information in the spreadsheet, namely, the combinations of ECDs that would cause task failure, was not carried over into this time-ordered task start-stop file. These ECDs were hardwired into the SBM itself in a C++ class named "Task." The "Task" class included a Boolean method or function for each task type and platform type pair that was evaluated when needed to determine whether a specific platform of that type with its then current set of applicable ECDs had the capability to perform that task.

G.2 Force Description/System Performance: Level 3

The next largest input file was one that supplied some of the needed information about the force played in the vignette. These file excerpts were presented in annotated form with any text following a "//" (including the double slash itself) treated as a comment. The first line gave the information described in the comment. To separate a comment from input data to the SBM, the comments have been converted to italics in this appendix.

// total # blue platforms + red units, # red units, index # of C2V, index # of Commander's vehicle,
// index # of higher echelon, index # of adjacent unit
31 6 6 7 23 24

The next entries in this file consist of data for each platform of the OWNFOR. The first example is for the first armored robotic vehicle (ARV):

// data for ARV #1
// name as string, platform type # (0=red unit, 1=arv, 2=uav, 3=c2v, 4=mcs, 5=nlos_c),
// index # (counting from 0, really redundant since should be I for Ith "mover" read in),
// side (0=OWNFOR, 1=OPFOR), max speed (in meters per second) that platform is capable of
// (fictitious value)
// ,
arv 1 0 0 25.
// # of nodes on path
7
// x-coordinate of first path node, y-coordinate of first path node, dwell time at first path node,
// departure speed in m/s from first path node
86300. 95000. 7200. 5.556
// same for second path node
86470. 95990. 0 1.389
86800. 97900. 0 1.389
86700. 99600. 0 1.389
87000. 101200. 0 1.389
87500. 102200. 0 1.389
// same for last path node except that -1 indicates dwell time is infinite
// (unless special event occurs)
88300. 103500. -1 1.389
// data for fuel consumption: capacity, starting level, consumption rate per km moved
// (all fictitious, but doesn't matter in this vignette)
200. 200. 2.0

// # of sensors onboard used for surveillance
3
// name of sensor
lras
// angular coverage in degrees, max range in m, min_acq_time in sec,
// acq_range_factor in sec/km, sense on move flag (not used), acq_rn_corr,
// night_capable_flag, range_los_parameter
// (all are fictitious values.)
*// mean time to acquire given target in coverage is min_acq_time + range **
// acq_range_factor/1000.0
// probability of having LOS to range (in meters) is exp(-range/range_los_parameter)
180. 10000. 10.0 40.0 1 1. 1 3000.
// name and same data for next sensor
day
180. 4000. 10.0 40.0 1 .5 0 2500.
// name and same data for third sensor on this platform
night
140. 3000. 10.0 50.0 1 .5 1 2500.
// number of shooters (negative # means a shooter is there for purposes of ECD bookkeeping
// but it is never used and has no characteristics
// a 0 means there isn't even a ECD shooter (e.g., on uav)
// a 1 or more means there is a real shooter that is played and has characteristics as described
// (see nlos_c example below)
-1
// number of ECD degradations in each category for this platform type
// categories are (in order) Mobility, Firepower, Target Acquisition, Surveillance,
// Communications, Survivability, Crew, Passengers, Other, Catastrophic
// As the model evolved, it became more convenient to allow space for the same number of
// degradations for all platform types
4 5 2 4 5 5 7 0 1 1
Next is first UAV
// data for UAV#1
// data for first UAV starts here
// max speed is surely wrong, but no speed close to that is ever demanded of it anyway
uav 2 3 0 25.
36
86500. 94500. 300. 16.667
88300. 101100. 0 16.667
86600. 95700. 0 16.667
87500. 104500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667

87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
85600. 104000. 0 16.667
88000. 100500. 0 16.667
87600. 105000. 0 16.667
// note that model logic generates a final UAV path node at the current location of its ground
// station after a specified mission duration
89000. 102500. -1 16.667
// fuel data
20. 20. 0.1
// sensor data again fictitious (range represents 3km slant range from 500 m)
// acoustic sensor never acquires things with enough precision to target them
// (it could be eliminated with no
// difference in outcome of this vignette in this model)
3
flir
360. 2996.13 10. 40. 1 1. 1 12500.
EO
360. 2996.13 10. 40. 1 1. 1 12500.
acoustic
360. 5000.00 10. 40. 1 0. 1 5000.
// no shooter on board UAV
0
// ECD degradations
4 5 2 4 5 5 7 0 1 1

The following example shows command and control vehicle (C2V) inputs:

```
// data for C2V (C2V #1 is the main C2V, C2V #2 is the commander's vehicle)
// platform name, platform type, index #, side, max_speed
c2v     3       6       0       25.
// path nodes
6
86300. 95000. 7200.   1.389
86800. 97900. 0       1.389
86700. 99600. 0       1.389
87000. 101200.        0       1.389
87500. 102200.        0       1.389
87300. 103200.        -1      1.389
// data for fuel consumption: max now rate
200.    200.    2.0
// sensors
2
day
180.    4000.   10.     40.     1       1.      0       2500.
night
140.    3000.   10.     50.     1       .5      1       2500.
// negative # indicates "virtual" shooter
-2
// # ECD degradations
4       5       2       4       5       5       7       0       1       1
```

The following example shows Mounted Combat System (MCS) inputs:

```
// data for MCS #1 of PLT 1
// MCS are played only as movers, they don't shoot, acquire, decide, have failures,
// suffer combat damage, or get repaired
// this is obviously something to improve in follow-on model
mcs     4       8       0       25.
6
86300.          95000. 7380.   1.389  // 180. seconds to allow ARV to get 1 km ahead of column
86800. 97900. 0       1.389
86700. 99600. 0.      1.389
87000. 101200.        0.      1.389
87500. 102200.        0.      1.389
88100. 103550. -1     1.389
200.    200.    2.0
// no sensors played on MCS
0
// no shooters played on MCS
0
// no ECD played for MCS
```

```
// data for NLOS-C #3
Data for NLOS-C #3
// data for NLOS-C #3
nlos_c 5        19      0       25.
10
93000. 85500. 3172.  2.778
91500. 86500. 0.     2.778
90000. 87000. 0.     2.778
88800. 88000. 0.     2.778
87000. 91000. 0.     2.778
85950. 92000. 0.     2.778
85550. 91900. -1.    2.778  // reach here at about 0354  (6 min. early)
85750. 91400. -1.    2.778  // move to this point after first fire mission
84950. 90500. -1.    2.778  // move to this point after second fire mission
84650. 91000. -1     2.778  // move to this point after third fire mission
// data for fuel consumption: max  now  rate
200.    200.    2.0
0
1
main_gun
30000.
2.09    6.26
1.05    3.14
24      24
20.0    10.0
1.      1.      1.      0
4       5       2       4       5       5       7       0       1       1
```

In addition to the real OWNFOR platforms, the model had two "virtual" OWNFOR platforms that were, in the end, only used as message sinks. In the earlier design of the model, it was anticipated that there would be a need to shift to alternative courses of action and possibly request help from an adjacent unit. Hence, there were two entities higher echelon ("HEch") and adjacent unit ("AdjUnit") for which data is entered:

```
// higher echelon is just a "dummy" unit created as a communications node with the intention of
// using it to implement alternative courses of action.  It could be eliminated with no effect on
// current vignette
// name, platform type (6=message_node), index #, side, max_speed
HEch  6       23      0       25.
2
84000. 93000.         0       5.
84100. 93000. -1      0
// data for fuel consumption: max  now  rate
200.    200.    2.0
0
0
0       0       0       0       0       0       0       0       0       0
```

```
// data for Adjacent Unit node
AdjUnit         6       24      0       25.
2
88000. 94000. 0         5.
89000. 96000. -1        0
// data for fuel consumption: max  now  rate
200.    200.    2.0
// no sensors
0
// no shooters
0
// no ECD played
0       0       0       0       0       0       0       0       0       0
```

For the OPFOR, each unit consisted of only one or two platforms and was treated as a circular area. Here is typical data for an OPFOR unit:

```
// data for Red Observation Post
// red units are played as collective "blobs" rather than discrete platforms
// this is something that you'll probably want to change
// name, type (0 = red unit), index #, side (1= OPFOR), max_speed
red_OP          0       25      1       25.
2
87700. 103600           -1.     5.556
89000. 102500.          -1.     0.0     // only moves to this path node when hit by rounds
// data for fuel consumption: max  now  rate
200.    200.    2.0
// no sensors
0
// no shooters
0
// no ECD played
0       0       0       0       0       0       0       0       0       0
// this red unit consists of 1 platform type
// located randomly within a 20 m circle centered at unit's current center
1       20.
// name of platform type, index # of target platform type, count of such platforms,
// probability of being acquired given in footprint
// of blue munition, probability of being killed given selected for attack from among
// acquired items in footprint
apc     0       1       .65     .8
// data for Red AT Co.
```

The SBM allowed polygonal areas to be defined as “no fire” zones. Any calls for fire into such a zone would be ignored. In the vignette for the demonstration, there was only one “no fire” zone. Its data is as follows:

// The vignette has one "no fire" zone defined as the inside of a rectangular region
1 *// # "no fire" zones*
4 *// # vertices in first no fire zone*
// The polygonal region's boundary is defined by connecting the vertices in the order their
// coordinates are read. To close the polygon, the last vertex is then connected to the first.
91000. 105000. *// coordinates of the vertices defining the rectangular region*
93000. 105000.
93000. 106500.
91000. 106500

Finally, the vignette orders each unmanned aerial vehicle (UAV) to return to its ground station before it would run out of fuel. Here is the data for the return to ground station events that are scheduled. Note that since the vignette ended at absolute time 36000 (1000 in military time), the third UAV is still airborne beyond the end of the vignette (unless it has been shot down or crashed because of a failure).

3 *// # of special external events to be read in at start of each replication*
// time event_type platform
// times are absolute times in seconds, not times after start of scenario
20520.0 15 3 *// uav #1 (platform #3) returns to ground station*
30780.0 15 4 *// uav #2 (platform #4) returns to ground station*
37800.0 15 5 *// uav #3 (platform #5) returns to ground station*

G.3 Communications Network Routing

The third input file used by the SBM contained the information on the communications network so that each message type was passed to the proper nodes. An excerpt of the file follows:

// For each message type, there is a table
// Read each line as follows: first field = message type, second field = sending node #,
// subsequent fields are nodes to which message is sent, -1 indicates end of line
// Situation Awareness messages
0 0 6 7 -1
0 1 6 7 -1
0 2 6 7 -1
0 3 6 7 -1
0 4 6 7 -1
0 5 6 7 -1
0 6 6 7 -1
0 7 6 7 -1
0 8 -1 // Nodes 8-16 are the MCS platforms for which no messages were generated.
0 9 -1
0 10 -1
0 11 -1
0 12 -1
0 13 -1
0 14 -1
0 15 -1

```
0       16      -1
0       17      6       7       -1
0       18      6       7       -1
0       19      6       7       -1
0       20      6       7       -1
0       21      6       7       -1
0       22      6       7       -1
0       23      23      -1
0       24      24      -1
// Common Operating Picture updates
1       6       0       1       2       3       4       5       7       17      18      19      20
        21      22      23      -1
1       7       0       1       2       3       4       5       6       17      18      19      20
        21      22      23      -1
1       23      6       7       24      -1
1       24      23      -1
// ECD status update
2       0       6       7       -1
2       1       6       7       -1
2       2       6       7       -1
2       3       6       7       -1
2       4       6       7       -1
2       5       6       7       -1
2       6       6       7       -1
2       7       6       7       -1
2       17      6       7       -1
2       18      6       7       -1
2       19      6       7       -1
2       20      6       7       -1
2       21      6       7       -1
2       22      6       7       -1
// activity update
3       0       2       3       -1
3       1       2       3       -1
3       2       2       3       -1
3       3       2       3       -1
3       4       2       3       -1
3       5       2       3       -1
3       6       6       -1
3       7       7       -1
// acquisition report
4       0       6       7       -1
4       1       6       7       -1
4       2       6       7       -1
4       3       6       7       -1
4       4       6       7       -1
```

```
4       5       6       7       -1
4       6       6       7       -1
4       7       6       7       -1
// call for fire
5       6       17      18      19      20      21      22      -1
5       7       17      18      19      20      21      22      -1
5       23      -1
5       24      -1
// fired "over"
6       17      6       7       -1
6       18      6       7       -1
6       19      6       7       -1
6       20      6       7       -1
6       21      6       7       -1
6       22      6       7       -1
// BDA_request message
7       2       0       1       -1
7       3       0       1       -1
// BDA_report
8       0       2       3       -1
8       1       2       3       -1
// cease fire order
9       6       17      18      19      20      21      22      -1
9       7       17      18      19      20      21      22      -1
// target update
10      0       6       7       -1
10      1       6       7       -1
10      2       6       7       -1
10      3       6       7       -1
10      4       6       7       -1
10      5       6       7       -1
10      6       6       7       -1
10      7       6       7       -1
// call for fire copy
11      6       7       -1
11      7       6       -1
// cannot fire
12      17      6       7       -1
12      18      6       7       -1
12      19      6       7       -1
12      20      6       7       -1
12      21      6       7       -1
12      22      6       7       -1
// fire mission done copy
13      6       7       -1
```

G.4 Miscellaneous Run Parameters

Another small input file was used to set the number of replications, random number seed, vignette start and stop (max) times, and the acquisition update interval. The file was used to set trace flags for whether additional model process information (mainly for debugging purposes) should be printed out—acquisition, communications, ECD, movement, fire mission, health bar, and task-resourcing processes, respectively. And finally, the times for night and day to start were set. In the following example case, a change from night to day would occur at 21600 (0600).

```
// #reps  rn_seed  time_start   time_max      acq_interval
10        5133771  7200.        36000.        50.0
// trace flags:
//ACQ  COMMO      ECD   MOVE        FM     BAR    TR
0       0       0       0       0       0       0
// day-night history
night   7200.   day     21600.
```

G.5 Capability State History: Level 3

This part of the SBM effort combined a fairly high-resolution representation of platform capability with a mostly fictitious representation of the component-level state changes. Output from the component state vector generator (CSVG),* the time-ordered sequence of component state vectors, was used as input to the MUVES $O_{2,3}$ mapper.† This mapping tool evaluated the fault trees of the corresponding platform to determine the current functional capability state. The resulting output, or capability state history, was a time-ordered sequence of state vectors where each time corresponded to a (potential)‡ change in the combination of capabilities (as discussed in appendix D) that applies to the platform at that time.

The capability state history was the main driver of the right side of the Missions and Means Framework diagram. From the user-selected mean time between combat damage events, mean number of components killed per such event, mean time between failure for each component, and mean time to repair (MTTR) each component, the model user can generate a script of the changes in functional capability for each platform during a replication (for as many replications desired) of the vignette. An excerpt of the script is shown in figure G-4.

*Details on the CSVG are discussed in section G-7.

†An AWK script was developed to read the CSVG output and make system calls to the MUVES O2,3 mapping tool each time a state vector was encountered. The script then formatted and output the results for each call in a single input file for the SBM.

‡A component-level state change generated by the CSVG may not, in some cases, result in changing the level-3 capabilities that apply to the platform. For example, failure of a redundant component may not change the capability if alternate path sets of the fault tree remain intact.

```
time =   7605.504  event type =  1 platform = 11  #dys comps =    1 damage vector =
SV: (0)  0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
SV: (1)  0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
SV: (2)
00000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000
00000000000000000000000000
 time =   7899.651  event type =  0 platform = 12  #dys comps =   10 damage vector =
SV: (0)  0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
SV: (1)  0 0 0 0 10 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0
SV: (2)
00000000000000000000000000000000000000000000000000000000000000000000000000000146a82c000000000000000000000
00000000000000000000000000
 time =   7937.119  event type =  2 platform = 12  #dys comps =    9 damage vector =
SV: (0)  0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
SV: (1)  0 0 0 0 9 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0
SV: (2)
00000000000000000000000000000000000000000000000000000000000000000000000000000146a824000000000000000000000
00000000000000000000000000
 time =   8141.230  event type =  1 platform =  0  #dys comps =    1 damage vector =
SV: (0)  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
SV: (1)  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
SV: (2)
00000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure G-4. Excerpt for time-ordered ECD change event file.

Each entry in this file contained four lines (though because of width of paper some are wrapped). The first line for each event began with the string "time =" followed by the time in seconds when the event causing the change in capability occurred. Then followed a number indicating the event type (0 = combat damage, 1 = failure, 2 = repair of a component). Next was the platform number as used in the SBM, and then a running count of the number of dysfunctional components on the platform as a result of this event. These entries just echoed the input from the CSVG. The final entry on the first line was the string "damage vector =" leading into the last three lines.

The second line had the heading "SV: (0)" and an entry of 0 or 1 for each ECD applicable to that platform type. If an ECD was triggered (i.e., the fault tree was cut), then this entry would be a 1; if the ECD was not triggered (i.e, the fault tree was not cut and the corresponding capability had not been lost), then the entry would be a 0. This was the heart of the $O_{2,3}$ output. The $O_{2,3}$ mapper involved the complex process of evaluating every applicable fault tree for each component-level state change event to generate the ECDs applicable to the platform.

The third line had the heading "SV: (1)" and an integer entry for each ECD applicable to that platform type. The integer was a count of the number of critical components currently dysfunctional on the fault tree for the corresponding ECD. This was not used in the SBM itself but was used as a check on the data generation process. For example, if the total of the integer

entries was less than the number of dysfunctional components on line 1, then something was wrong with the process. Similarly, if an ECD had been triggered but the count of dysfunctional components on its fault tree was 0, then another type of error had occurred.

The fourth line had the heading "SV: (2)" and was simply a copy of the component (level 2) hexadecimal damage vector that the $O_{2,3}$ mapper evaluated to get the ECDs on line 2. This echoing of the level-2 vector was a late addition to the file format and was used to collect statistics on the correlation of component damage to ECDs and task failures.

G.6 Merged File of External Events

The two files of time-ordered events—the first being the file of task start/stop events, as illustrated in figure G-5, and the second being the file of ECD change events—were then merged with time order maintained. This was a relatively straightforward operation with some minor complications caused by the fact that the second file may contain more than one replication of the ECD change events for the vignette and by the nonuniform length of the information for each event type. However, it had been successfully implemented in a utility C++ program.

This file consisted of an entry for each event. Each line began with a time representing a new event. An excerpt from the resulting merged time-ordered event file is shown in figure G-5.

```
8160.6030 1 6 3 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000008000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
 8584.3940 2 21 5 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 8 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
000000000000000000000000000000000000000000000000000000000000000000000146a8040000000000000000000
0000000000000000000000000
 8655.9610 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
 8961.1680 2 21 5 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
000000000000000000000000000000000000000000000000000000000000000000000146a8000000000000000000000
0000000000000000000000000
 9480.0000 4 3 2 12 UAV1 Conduct_Surveillance_ART1.3.4
 9480.0000 4 4 2 12 UAV2 Conduct_Surveillance_ART1.3.4
 9554.7920 1 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
00000000000000000000000000000000000000000000000004000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
 9600.0000 5 3 2 10 UAV1 Conduct_Tactical_Reconnaissance_ART1.3.3
 9662.1700 2 21 5 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 2 2 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
000000000000000000000000000000000000000000000000000000000000000000000106a8000000000000000000000
0000000000000000000000000
 9665.5070 2 6 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure G-5. Excerpt of merged time-ordered event file.

The first event in this excerpt occurred 8160.6030 s after midnight (first field) in the vignette and consisted of a failure (the "1" in the second field) on the C2V (third field platform 6) of type C2V (fourth field value of "3") resulting in the ECDs indicated by the next 34 entries of "0" or "1." In this example, only the fifth ECD was triggered by this failure, and from the next group of 34 entries of "0" or "1," only one component in the fifth ECD was currently dysfunctional. Finally, one can look at the bit string and count over to where the "8" is to determine which component on the platform actually was dysfunctional (the bit string should be converted from hexadecimal to binary to get the actual component number).

The second event in this excerpt occurred 8584.3940 s after midnight and consisted of a repair (the "2" in the second field) to platform number 21, which was of type 5 (an NLOS-C). The 0 and 1 entries after that indicate the ECDs and component counts following the repair. The hexadecimal bit string can be used to find the numbers of the components currently dysfunctional.

The third event at 8655.9610 was a repair ("2") to platform 0, which was an ARV ("1"). The 0 and 1 entries again indicated the ECDs in effect and the number of critical components dysfunctional per ECD. The hexadecimal bit string permitted one to identify the actual components currently dysfunctional on the platform.

The fourth event at 8961.1680 was again a repair to platform 21, an NLOS-C (type 5). ECD, count of dysfunctional components, and index numbers of dysfunctional components were available as before.

The fifth event at time 9480.0000 was the start of a task* (indicated by "4" in the second field) by platform 3 (third field) of type UAV ("2" in the fourth field). The task type was 12. The alphanumeric fields simply confirmed the identification of the platform as UAV 1 and described the task. The actual check as to whether UAV 1 currently had the capability to perform the task was accomplished by a function of the "Task" class that evaluated the current ECDs of UAV 1 against the task requirements; that evaluation function was hard-wired in the SBM code.

The sixth event occurred at time 9480.0000 and was similar to the fifth event except that it applied to UAV 2.

The seventh event occurred at time 9554.7920 and consisted of a failure event ("1" in field 2) on platform 2 (field 3), which was one of the ARVs (field 4). From the "1" in the first group of 34 "0" or "1" entries, this was an M1 ECD. The next set of 34 binary entries indicated that one critical component for M1 was dysfunctional and one for M4 was dysfunctional. The hexadecimal bit string allowed one to determine exactly which component(s) were now dysfunctional.

*More precisely, it was the beginning of an interval during which the platform should have the capability needed to perform the task.

The eighth event occurred at time 9600.000 and consisted of the end of a task (the "5" in second field) for platform 3, which was a UAV, and the task to end was a type-10 task. The alphanumeric identifier and string were as before.

The last two events in this excerpt were both repair events.

By running through this time-ordered event file, the SBM executed the major events of the vignette, namely, the beginning and end of tasks and the changes in platform state (and hence capabilities). There were other events that were dynamically generated on the fly within the model (described in appendix F).

G.7 Damage Component State History (Level 2)

Because of the one-sided nature of the model, the component state changes to OWNFOR platforms were not dynamically generated but rather scripted and driven from an external event file.* The state changes could be due to combat damage, part failure, repair, or resupply.

A pair of programs for preprocessing was used to generate this event file. The first was a level-2 CSVG, and the second was the MUVES $O_{2,3}$ mapper program, the latter of which determined the system capability following a component state change.

At level 2 (components, forces), a simple event-sequenced model† was written to generate a script or history of the component‡ state changes for each platform over a specified number of replications of the vignette. The script required the following input data for execution:

- A user-specified time between combat damage events, a mean number of components killed per such event, and a mean time between component failures (MTBF).§
- An MTTR each component.**
- The number of components for each platform.
- The number of platforms of each type.
- Random number seed.
- The number of vignette replications to be generated.

---

*Not to be confused with the TOEL supplied by DRC as a vignette description and from which the time-ordered task start/stop time file was developed.

†A program of about 600 lines (excluding header files) written in the C programming language and referred to as the component state vector generator (CSVG).

‡For this purpose, a component means a critical component, i.e., a component that is relevant to at least one system-functional attribute. A component is irrelevant to a function if its state (whether functional or dysfunctional) never affects whether the attribute fault tree is cut.

§This is the time between failures to be used for generating failure events for each component on a platform, not the time between the event that a component on the platform failed. The former is N× the latter if the platform has N components (and all have the same MTBF).

**One elementary improvement that could be made to this model would be to use a different MTBF and MTTR for each component type. This was not done since all of the data were fictitious for demonstration purposes.

The script output was a time-ordered sequence of random events (using an exponential distribution of inter-event times for each event type). Each time a combat damage event occurred, not only was the next combat damage event for that platform scheduled using the exponential inter-event time, but a set of damaged components was generated* and a repair event for each damaged component was scheduled using the MTTR as the parameter of another exponential random number draw.†

Each time a wear-and-tear‡ failure event occurred for a component on a platform, it was toggled to a dysfunctional state. Then a repair event for that component was scheduled by drawing a repair delay time from the exponential distribution with MTTR and adding it to the current time in the model.

Finally, each time a repair event occurred, the component in question was toggled back into functional state (from its previously dysfunctional one).§

In figure 1 of the main text, the model is referred to as the component status vector generator. The CSVG's primary output is a time-ordered sequence of component state vectors** where each time corresponds to a change in the vector whose values are the 0s and 1s indicating the functional/dysfunctional states of the corresponding components on the platform. An excerpt of the program output is shown in figure G-6.

*The number of components killed per combat damage event was generated using a truncated geometric distribution having the mean value input by the user. It was truncated because it had both a lower bound (at least one component was killed or it would not have been a combat damage event) and an upper bound (no more components could be killed than were present on the platform). The particular set of components to be killed (rendered dysfunctional) was chosen by picking one component at random on the platform and then for each additional component to be killed as a result of that event, picking another component using an algorithm designed to achieve a clustering of the component index numbers.

†An exception to this procedure occurred when a component selected to be killed was already dysfunctional (and not yet repaired) as a result of previous combat damage or a wear-and-tear failure. In this case, its scheduled repair time was left unchanged.

‡"Wear-and-tear" failure is the generic term used here for all causes of component dysfunction other than combat damage from enemy weapons. This includes not only failures caused by wearing out components, but also failures caused by simple aging, even of components that are never used.

§There is an exception in the case of a component that has suffered both a wear-and-tear failure and been rendered dysfunctional by combat damage. In this case, if there is a later repair time scheduled on the event queue, the component is not toggled from dysfunctional to functional at that time, but rather is left dysfunctional until the latest repair time on the event queue. This exception is designed to prevent additional combat damage from speeding up a component's repair.

**Sometimes referred to as damage vectors.

```
time =  7605.504  event type =  1 platform = 11  #dys comps =   1 damage vector =
00000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000
00000000000000000000000000
time =  7899.651  event type =  0 platform = 12  #dys comps =  10 damage vector =
00000000000000000000000000000000000000000000000000000000000000000000000000146a82c000000000000000000000000
00000000000000000000000000
time =  7937.119  event type =  2 platform = 12  #dys comps =   9 damage vector =
00000000000000000000000000000000000000000000000000000000000000000000000000146a824000000000000000000000000
00000000000000000000000000
time =  8141.230  event type =  1 platform =  0  #dys comps =   1 damage vector =
00000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
time =  8160.603  event type =  1 platform =  6  #dys comps =   1 damage vector =
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
time =  8584.394  event type =  2 platform = 12  #dys comps =   8 damage vector =
00000000000000000000000000000000000000000000000000000000000000000000000000146a804000000000000000000000000
00000000000000000000000000
time =  8655.961  event type =  2 platform =  0  #dys comps =   0 damage vector =
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure G-6. Level-2 state vector change event file.

Each line of the file (spread across multiple lines in this excerpt) corresponds to one state change event. The first entry on each line is the time in seconds at which the state change occurred, the second entry is an integer indicating the event type (0 = combat damage, 1 = wear-and-tear failure, 2 = repair), the third entry is the platform number, the fourth entry is the current count of dysfunctional components on the platform, and the fifth entry (usually spread over multiple lines in the format of this report) is the actual component-level state vector in hexadecimal. When the

hexadecimal is converted to binary, a "1" in the nth position will indicate that the nth component on that platform is dysfunctional, and a "0" in the nth position will indicate that the nth component is still (or again) functional.

G.8 Task to Capabilities ($O_{3,4}$)

Army-operational SMEs* reviewed the 31 tasks in the vignette and the functional capabilities of each platform. The review identified system-specific functions† that, if degraded during the vignette simulation, would cause platform task failure. Input files were not used to correlate task requirements to platform capabilities ($O_{3,4}$). Evaluation of the task fault trees was hardwired in the SBM as an object-oriented method.‡

The designers of the SBM decided that rather than read in and parse the fault tree for each task for each degradation category and evaluate whether it was cut or intact at run time, the lower-risk (but less flexible) approach would be simply to hardwire the fault trees into the SBM code and have them available as functions (object-oriented methods) to be evaluated to determine whether their corresponding fault trees were cut. There was a "Task" class defined in the SBM that included a virtual Boolean "pass/fail" function. The function call required the ECDs describing a platform's current state. The function would then return a true value (pass) if the platform were capable of performing the task or a false value (fail) if the platform did not have the capability to perform the task.

*Based on the standard tasks lists, such as the AUTL, UJTL, and lead system integrator (LSI) SITL.

†Platform functional definitions for the MMF demonstration are described in appendix D.

‡The fault trees were translated into Boolean expressions in the C++ code for the functions. Then the true-false value corresponding to each function in the current platform state was substituted into the expression via function arguments. The value of the expression then indicated whether the fault tree for the task of interest was cut or intact.

Appendix H. Model Outputs

There were five text files generated by the Storyboard Model (SBM): the log file, the event history file, the health history file, the task history file, and the correlation file. The log file was intended primarily to be a debugging aid for the developer and consisted of a record of the events that occurred in a run of the model. The event history file was used to drive the situation map display by which one could see how the platforms and units move, message traffic, acquisitions, and firing events; consequences of some failures, damage, and repairs were also evident. The health history file was used to drive the health bar display. The display showed current task demands of each platform and the capability available to perform. The task history file was intended mainly to guide analysts in determining which task failures were traceable to various component losses. Finally, the correlation file reported statistics based on the results of the run.

H.1 Log File

The log file was written to provide information for model development—to monitor the computer program and verify it was running correctly. It was not intended to be a file containing results of interest to the general analyst, although some of the output might interest others than the developer.

Figure H-1 displays what the first section of the log file typically looked like. This first section provided the date and time the model run was initiated. Then it noted that the program was in the "get_run_parameters" function and has opened a file (a file named rpdrc.nc in this example) for reading. It then echoed (with labeling) the values it found in that file. In this example, the values are 10 for the number of replications of the vignette to be run (n_rep), 5133771 for the initial value of the random number seed (rn_seed), 7200.000000 for the time at which the vignette started (time_start), 36000.000000 for the time at which the vignette ended (time_max), and 50.000000 for the interval between acquisition checks by the sensors (acq_interval). The last three of these values are in seconds (seconds after midnight in the case of time_start and time_max).

```
Fri Mar 11 08:43:17 EST 2005
In get_run_parameters.  Opened file rpdrc.nc for reading
    n_rep = 10  rn_seed = 5133771  time_start = 7200.000000 time_max = 36000.000000
    acq_interval = 50.000000
    ACQ_TRACE = 0  COMMO_TRACE = 0  DCS_TRACE = 0  MOVE_TRACE = 0  FM_TRACE = 0
BAR_TRACE = 0  TR_TRACE = 0
 ********************************************
```

Figure H-1. First section of SBM log file.

The remainder of this section reported the status of various program trace flags. In this example, all flags were set to zero; however, it was possible to set one or more trace flags to one and thus cause more trace information to be written to the log file. One would ordinarily run the program with these trace options turned off, as the information they provide was intended for even more specialized debugging purposes than the general log file. However, if desired, the user could choose to print additional information relevant to the target acquisition process, the communications process, the functional capability representation (or elements of capability degradation [ECDs]), the movement process, the fire mission process, the health bar information, and the task requirements modeling by setting the corresponding trace flags.

Next, the program performed some initialization steps for the replication of the vignette that was about to begin. The initialization steps were repeated for each replication in a given program run. This allowed the runs to generate independent, identically distributed random outcomes.* Figure H-2 is an example printout during this initialization.

```
In initialize_rep. On replication no. 1
  event_schedule Day_Night_change. tm=21600.000000 et=12  …     event_schedule exits
  event_schedule risk table change. tm=14400.000000 et=16 …     event_schedule exits
  event_schedule risk table change. tm=21600.000000 et=16 …     event_schedule exits
initialize_task_bar_info called with 6 and 31
in get_movers. opening file pathdrc2.nc
n_movers=31 n_blue_plt=25 n_red_units=6 C2V=6 ALT=7 HEch=23 AdjUnit=24
```

Figure H-2. SBM log file: replication initialization.

The first line indicates that the program had called the function "initialize_rep" for replication number 1. The next three lines report that three events had been scheduled. The first event consisted of toggling the day-night status at 21600 s after midnight (0600), which was an event of type 12. The second event consisted of changing the risk table (an event of type 16) from the values used in the initial phase of the mission to those used in the second phase, and this was scheduled to occur 14400 s after midnight (0400). The third event was again a type-12 event (changing risk table for the next phase of the battle), and it was scheduled to occur at time 21600 s after midnight (0600). These would be the first three events placed on the event queue (in order of the times they were to be executed [14400, 21600, and 21600]; in the case of the last two there was a tie, so the first one of those two scheduled would be executed first). Of course, there were likely to be many other events scheduled and executed long before the simulation time reached 14400 or 21600, but these events were placed on the queue to be dealt with when the time came. The ellipses represent parameters that were not relevant to these event types.

*One of the verification steps was to check that no differences in initial conditions were carried over from one replication to another, except for being at a different point in the random number stream.

Next, the vignette engine called the function “get_movers”, which read in most of the data describing the own force (OWNFOR) platforms and opposing force (OPFOR) units in the vignette. That function opened a file (named, in this example, “pathdrc2.nc”) containing the data to be read. The last line echoed some values to be used in the vignette. They indicate, respectively, that there were 31 potential movers (OWNFOR platforms and OPFOR units) in the vignette, that 25 were blue (OWNFOR) platforms, 6 were RED (OPFOR) units, mover no. 6 was the C2V, mover no. 7 was the alternative (backup C2V), mover no. 23 was the message sink for the OWNFOR Higher Echelon, and mover no. 24 was the message sink for OWNFOR Adjacent Unit.

Next, the program read in the data for each OWNFOR platform and each OPFOR unit (the data described in appendix G, section G.2). The program initialized some of the variables used to track the functional capabilities (or ECD) and platform “health” for task execution. The data and the results of the initialization were echoed to the log file. Figure H-3 is an example of the information printed for one of the platforms.

```
Now reading data for platform name arv ptype 1 index 0  side 0 with max_speed = 25.000000
 new path with 7 nodes allocated
 node 0  x,y = (86300.000,95000.000) dwell time = 7200.000  max dep speed = 5.556
 node 1  x,y = (86470.000,95990.000) dwell time = 0.000  max dep speed = 1.389
 node 2  x,y = (86800.000,97900.000) dwell time = 0.000  max dep speed = 1.389
 node 3  x,y = (86700.000,99600.000) dwell time = 0.000  max dep speed = 1.389
 node 4  x,y = (87000.000,101200.000) dwell time = 0.000  max dep speed = 1.389
 node 5  x,y = (87500.000,102200.000) dwell time = 0.000  max dep speed = 1.389
 node 6  x,y = (88300.000,103500.000) dwell time = 28801.000  max dep speed = 1.389
fuel: capacity = 200.000  onboard now = 200.000  LperKm = 2.000
   3 sensor types
 sensor lras :ang, rng, rate_min, rate_range,move, night, range_parameter = 180.000  10000.000
10.000  40.000  1 1.000 1 3000.000
 sensor day :ang, rng, rate_min, rate_range,move, night, range_parameter = 180.000  4000.000
10.000  40.000  1 0.500 0 2500.000
 sensor night :ang, rng, rate_min, rate_range,move, night, range_parameter = 140.000  3000.000
10.000  50.000  1 0.500 1 2500.000
   -1 shooter types
 reading info on ECD categories: m1 m2 m3 m4 f5 a1 a2 z1 z2 z3 x1 x5 s3 k1
 ECD_vector_print:  /M 0 0 0 0 /F - - - - 0 /A 0 0 /Z 0 0 0 - /X 0 - - - 0 /S - - 0 - - /C - - - - - - - /P *
/O - /K 0/
```

Figure H-3. SBM log file: path nodes, sensor parameters, and ECD.

The excerpt first identified the platform name (arv in this case), platform-type index (1), platform index (0), and side (0 indicating OWNFOR). It also echoed the maximum platform speed (25 m/s) on this line.

Next, the program logs the platform's path. This example indicates that the path has seven nodes, and for each node, it printed out the x,y coordinates (in meters from the vignette origin), the dwell time it was to spend at that node, and the departure speed from that node (1.389 m/s, which was ~5 km/h). The fuel parameters, fuel tank capacity, initial fuel onboard, and burn rate (liters per kilometer) were also echoed. These fuel parameters were totally fictitious, but they did not make any difference in this vignette as none of the platforms traveled far enough to run out of fuel, even if the correct tank capacities and burn rates had been used.

Following the speed, path, and fuel parameters (mobility-related data) just described, the program documents the sensor data. In this example, there were three sensors named long range acquisition system (lras), day, and night on the platform. Each sensor had the following characteristic details (shown in figure H-3):

- The angular coverage of the sensor (in degrees).
- The range of the sensor in meters.
- The minimum mean time to acquire a target in its field of view (in seconds).
- The increase in mean time to acquire a target in its field of view (in seconds per kilometer) for each kilometer of range from sensor to target.
- A Boolean flag to indicate whether the sensor could operate on the move (the value 1 in these examples indicated that these sensors could operate on the move).
- The dependence weight.
- A Boolean flag to indicate whether the sensor could operate at night.
- The P_{∞} mean line-of-sight range for the sensor.

Following the sensor parameters was a section echoing the firepower ("shooter") parameters. In this example, an Armored Robotic Vehicle (ARV), the only weapon was a machine gun that was not used in the scenario. A special value of –1 was read (and echoed) to indicate that there was 1 weapon on board, but no parameters were supplied for it. In this vignette, only the Non-Line-of-Sight Cannon (NLOS-C) platforms had non-trivial shooter data.

After the shooter data (if any) was read, the program read and echoed the ECD data pertaining to the platform. In the figure H-3 example, the ECDs for the platform type were m1, m2, m3, m4, f5, a1, a2, z1, z2, z3, x1, x5, s3, and x1. The platform was then initialized to the state indicated in the line beginning "ECD_vector_print:" This line had a field (character) for every ECD used in this vignette. For the ECD used for this platform, there would be a 1 if it is initialized with that degradation and a 0 if it did not have that degradation in effect (but may be so degraded later

in the play of the vignette). A hyphen indicated that the corresponding degradation was not used for the platform type, and an asterisk indicated that the degradation was not used at all (for any platform type) in the vignette.*

The program used mathematical objects called partially ordered sets† to assess the ECD and later use in the platform health bar display (discussed in section H.3). For each ECD category (mobility, firepower, acquisition, surveillance, communications, survivability, crew, passenger, other, and catastrophic), there would be at most 2^N possible combinations, where N is the number of degradation elements. In the example excerpted in figure H-4, the mobility category was used. It had 4 ECD elements (m1, m2, m3, and m4) yielding 16 possible combinations, which are numbered from 0 to 15.‡

```
Bar Mobility object created
 at bar section of Mover::read with k = 0 Mobility
 Bar fill_in_tier() & implications() set tier and imp_BP arrays:
   n_tiers = 5
   0 bitstring: 00000000000000000000000000000000 with tier: 0 & implied bitstring:
00000000000000000000000000000000 with tier: 0

   1 bitstring: 00000000000000000000000000000001 with tier: 1 & implied bitstring:
00000000000000000000000000000001 with tier: 1
              M1                                          M1
   2 bitstring: 00000000000000000000000000000010 with tier: 1 & implied bitstring:
00000000000000000000000000000010 with tier: 1
              M2                                          M2
   3 bitstring: 00000000000000000000000000000011 with tier: 2 & implied bitstring:
00000000000000000000000000000011 with tier: 2
              M1 M2                                       M1 M2
   4 bitstring: 00000000000000000000000000000100 with tier: 1 & implied bitstring:
00000000000000000000000000000100 with tier: 1
              M3                                          M3
   5 bitstring: 00000000000000000000000000000101 with tier: 2 & implied bitstring:
00000000000000000000000000000101 with tier: 2

              M1 M3                                       M1 M3

   6 bitstring: 00000000000000000000000000000110 with tier: 2 & implied bitstring:
00000000000000000000000000000110 with tier: 2

              M2 M3                                       M2 M3

   7 bitstring: 00000000000000000000000000000111 with tier: 3 & implied bitstring:
00000000000000000000000000000111 with tier: 3

              M1 M2 M3                                    M1 M2 M3

   8 bitstring: 00000000000000000000000000001000 with tier: 1 & implied bitstring:
00000000000000000000000000001011 with tier: 3

              M4                                          M1 M2 M4
```

Figure H-4. SBM log file: possible combinations of mobility ECDs for constructing the health display.

*In this vignette none of the OWNFOR platforms carried passengers, so there was no play of degradations resulting when one or more passengers were injured or killed.

†Details on the approach are discussed in appendix I.

‡There was a natural mapping between the binary representation of the integers from 0 to 15 and the bit strings indicating which ECDs were included in a given state.

After printing header information indicating which category (mobility) was being processed, the program printed out the number of tiers in the Haase diagram* followed by two lines for each combination. The first line starts with the integer value for that combination, then the word "bitstring" followed by the representation of that integer in binary with the tier number (which is nothing more than a count of the number of 1s in its binary representation). The words "implied bitstring" are then followed by the bit string representing the actual ECD triggered by the combination of elements (necessary since sometimes one or more ECDs imply others), and finally the tier number in the Haase diagram for the implied bit pattern. Below this line was a more reader-friendly representation than the bit strings; the actual ECD of the combination and the corresponding implied combination were listed. For example, the integer 8 corresponded to a binary bit string 1000,† which means ECD M4 was in effect. However, M4 was the total loss of mobility, which certainly implied both M1 (reduced maximum speed) and M2 (reduced maneuverability). Therefore, the state with degradation M4 implied the state with the set {M1, M2, M4} of degradations (which has the bit pattern 1011). At this point in the program, the information was merely stored for later use. Similar implied bit strings and Haase tiers were generated for the other ECD categories pertaining to the platform.

After generating the possible ECD combinations in each category, their implied states, and tiers, the program compared the states to the requirements for each task that may be encountered in the vignette. The program assigned a color (green, yellow,‡ or red) to each tier indicating whether the states at that tier always, sometimes, or never have enough capability in that category to perform the task. The results of this assignment were then printed as shown in table H-1.

Table H-1. SBM log file: color scheme for an example task and ECD category.

Task no. 9 LSI A4.2.1 Conduct Tactical Reconnaissance				
Mobility				
0	00000000000000000000000000000000	00000000000000000000000000000000	0	green
1	00000000000000000000000000000001	00000000000000000000000000000001	1	amber
2	00000000000000000000000000000010	00000000000000000000000000000010	1	amber
3	00000000000000000000000000000011	00000000000000000000000000000011	2	red
4	00000000000000000000000000000100	00000000000000000000000000000100	1	amber
5	00000000000000000000000000000101	00000000000000000000000000000101	2	red
6	00000000000000000000000000000110	00000000000000000000000000000110	2	red
7	00000000000000000000000000000111	00000000000000000000000000000111	3	red
8	00000000000000000000000000001000	00000000000000000000000000001000	3	red
9	00000000000000000000000000001001	00000000000000000000000000001001	3	red
10	00000000000000000000000000001010	00000000000000000000000000001010	3	red
11	00000000000000000000000000001011	00000000000000000000000000001011	3	red
12	00000000000000000000000000001100	00000000000000000000000000001100	4	red
13	00000000000000000000000000001101	00000000000000000000000000001101	4	red
14	00000000000000000000000000001110	00000000000000000000000000001110	4	red
15	00000000000000000000000000001111	00000000000000000000000000001111	4	red

*Haase diagrams are discussed in appendix I.

†Only the last four bits are shown here since they were the only ones that were needed with four ECDs.

‡Yellow is synonymous with amber in the Mission and Means Framework (MMF) literature.

The results in table H-1 were generated only for those tasks (of the 31 tasks defined and used in the demonstration vignette) that applied to the platform type. The program printed the task number and descriptive string. Then for each ECD category (mobility is again the example), it printed out the bit strings for the possible combinations of ECDs in that category, the corresponding implied bit strings (e.g., note that 1000 implies 1011), the tier of the implied bit string, and now the color of a platform at the same tier as that implied bit string when attempting to perform the task. This information was stored and used in generating the health bar or health meter display (discussed in section H.3).

Since this assignment of colors to the tiers would be the same for every platform of a given type, it was only generated for the first platform of each type in the vignette. Other platforms of the same type used the same stored mapping of task-tier pairings to colors.

After generating the mapping from platform ECDs and tasks to colors, the program performed a number of initialization tasks for the current platform. As shown in figure H-5, it initialized the platform as a mover, initialized the task pass-fail information for the platform, and then scheduled the platform departure from its first node at the appropriate time (14400 s after midnight in this case).

```
 Called Mover initialize
 initializing task p/f info for platform 0
 Mover::initialize scheduled departure for platform 0 at 14400.000000

 event_schedule  Move depart node. tm=14400.000000 et=0 pID=0 pID2=0 rID=0 f=0.000000
eq=0 mt=0 non-message event ct=0 mptr=0     event_schedule exits

event_schedule Send Message. tm=7200.000000 et=4 pID=0 pID2=6 rID=0 f=0.000000 eq=0 mt=2
ECD_status_update ct=0 mptr=0     event_schedule exits

Platform Name = arv
  platform type = 1
  mover index = 0
  time_last_loc_update = 7200
  then current position x, y = (86300,95000)
  last node # reached = 0
  not yet at final destination
  maximum speed undamaged = 25
  current velocity = (0,0)
  activity vector = 00000000000000000000000000000000
```

Figure H-5. SBM log file: initializing platform schedule of moves.

The program then scheduled an ECD update message to be sent to the command and control vehicle (C2V) at time 7200 s after midnight (which happened to be the start of the vignette). Then it echoed some of the initialization information, such as platform name, platform type, its index as a mover, the time its location was last updated, the position at that time, the number of the last path node it has reached, whether or not it was at its final destination, its maximum speed, its current velocity (as a two-dimensional vector), and a bit string activity vector (which was an unused remnant).

Next, the program reported the initial condition and capabilities of the platform in a more reader-friendly summary, as shown in figure H-6. Since the platform was initialized in pristine condition, there should be no degradations at this point. At later times in the program execution, this same summary was printed and may then show some capability degradations. Note the default condition was for the capability to be "up" (functional), not lost, and alive; this would show up in the printout even for capabilities not applicable to the particular platform type. (For example, this summary shows the crew members all alive even though an ARV has no crew on board. The reader should ignore those capabilities that do not apply.)

```
Degradations:
   M (Mobility):
     max speed possible now = 25       max speed factor = 1
     maneuverability factor = 1
     total_immobilization = no
     stop after time t scheduled = no
     stop after time t executed = no
     fuel onboard now = 200 liters
   F (Firepower):
     Total firepower loss ? no
   A (Acquisition):
    day sight lost? no
    night sight lost? no
   Z (Surveillance): F
    Sensor # 0
     current range = 10000
    Sensor # 1
     current range = 0
    Sensor # 2
     current range = 3000
   X (Communications):
     data commo up
     voice commo up
   S (Survivability measures):
     NBC protection is up
     Obscurant capability is up
     Silent Watch capability is up
      APS lost? no
      Secondary armament lost? no
   C (Crew):
     Commander is  alive
     Driver is  alive
     Squad leader is  alive
     operator 1 is  alive
      operator 2 is  alive
     uav recovery capability is up
   P (Passengers): not played
   O (Other): SA is up
   K (Catastrophic): platform K-killed ? no
  Estimated arrival time = 36001.000
```

Figure H-6. SBM log file: echoing initial condition of platform in human readable format.

The final initialization step for the platform was to calculate its estimated arrival time at its next node. For a platform that began the vignette with a positive dwell time at its first node, this estimated time of arrival was arbitrarily set to be 1 s beyond the end of the vignette. In either case, it was printed out as the final line of this section of the log file.

The program then read in the data for the next OWNFOR platform and performed similar initialization steps for it. The only major variation on what was just described is when the program encountered a platform that was a shooter (had weapons on board that were actually played in the vignette). In such a case, there was additional data read in and echoed for the platform, as shown in figure H-7. This data was read in right after the sensor data for the platform.

```
1 shooter types
main_gun
 mpi defl and range = (2.090000,6.260000)
 prec defl and range = (1.050000,3.140000)
 rounds: basic load 24   onboard now 24
 times: 1st round = 20.000000  between rounds = 10.000000
 Initial degradation factors = 1.000000 1.000000 1.000000 0
```

Figure H-7. SBM log file: echoing shooter data.

The first line indicated the number of shooters on the platform (one in this case) and the name of the shooter ("main_gun" in this case). Next, the program read in and echoed the mean point of impact errors in deflection and range and the precision errors in deflection and range; these errors were standard deviations in mils.* The program then read in and echoed the basic ammunition load that the platform carried and the amount actually on board at the start of the vignette (the number of rounds of the single type played). It did the same for the time in seconds to fire the first round of a fire mission and the time in seconds between subsequent rounds of a fire mission. The last line printed out the initial values for degradation multipliers for delivery errors (when the shooter suffered F2 degradation) and for, respectively, time to fire the first round and time between subsequent rounds (when the shooter suffered F3 or F4 degradation). The final zero on that line was the initial value of a Boolean flag indicating whether F5 degradation (total loss of firepower) was in effect; it was initialized to false (0) to indicate that such degradation has not yet happened.

This completes reading in, initializing, and echoing the initial conditions on the OWNFOR. A similar set of data was read in and echoed for each OPFOR unit. The echoing of the data was similar to what was described previously, although some of the information was irrelevant. For

*They were fictitious values chosen to yield a combined circular error probability of a reasonable magnitude for a modern cannon system. Indeed all of these performance parameters for shooters were fictitious, though reasonable for such a system.

example, the initial condition of the platforms showed all functions operational at the outset; however, the model did not play ECDs for the OPFOR, so one should not infer from the printout that ECDs were used. For the OPFOR, either a platform was fully functional (no degradation) or killed (all capabilities lost). Some parameters were unique to the OPFOR, an example of which is in figure H-8.

```
n_platform_types = 1  tgt_radius = 45.000000
 Red unit red_P_B
 Red force consists of 1 platform types in circle of radius 45.000000
 apc 2 0 0.650000 0.800000
```

Figure H-8. SBM log file: echoing red unit data.

The OPFOR units were represented as collections of platforms randomly (uniformly) distributed inside a circular disk of specified radius (and centered at the coordinates of its then current location which, like OWNFOR platform location, was modeled based on a sequence of path nodes with dwell times and departure speeds). The first line echoed simply gave the number of different platform types in the OPFOR unit and the radius over which they were distributed. The second line gave an abbreviated name for the OPFOR unit (in this case, it is "red_P_B" for red Platoon B). The third line repeated the information from the first. The fourth line gave, respectively, the name of platform type ("apc"), the initial number of such platforms in the unit (two), the number of such platforms that are dead (zero initially), the probability that a platform of this type would be acquired by the OWNFOR smart submunition if it were in the area scanned by the seeker, and finally, the probability that a platform of that type would be killed if it were acquired and selected for attack by the submunition.

The program next initialized the acquisition process for each OPFOR unit and each sensor on each OWNFOR platform. It initialized the random number to be used in the acquisition process (described in appendix E, section E.2), then it scheduled the first acquisition check event for the given sensor against each OPFOR unit. Sensor information is shown in figure H-9.

```
in acq_initialize.
 in acq_init with range = 4000.000000
 acquisition rn for this sensor 1 target 0 pairing set to = 0.226461
 acquisition rn for this sensor 1 target 1 pairing set to = 0.226620
 acquisition rn for this sensor 1 target 2 pairing set to = 0.115910
 acquisition rn for this sensor 1 target 3 pairing set to = 0.883979
 acquisition rn for this sensor 1 target 4 pairing set to = 0.562124
 acquisition rn for this sensor 1 target 5 pairing set to = 0.611192
  event_schedule ACQ check. tm=21600.000000 et=3 pID=2 pID2=0 rID=25 f=4000.000000 eq=1
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule ACQ check. tm=21600.000000 et=3 pID=2 pID2=0 rID=26 f=4000.000000 eq=1
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule ACQ check. tm=21600.000000 et=3 pID=2 pID2=0 rID=27 f=4000.000000 eq=1
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule ACQ check. tm=21600.000000 et=3 pID=2 pID2=0 rID=28 f=4000.000000 eq=1
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule ACQ check. tm=21600.000000 et=3 pID=2 pID2=0 rID=29 f=4000.000000 eq=1
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule ACQ check. tm=21600.000000 et=3 pID=2 pID2=0 rID=30 f=4000.000000 eq=1
mt=0 non-message event ct=0 mptr=0     event_schedule exits
```

Figure H-9. SBM log file: echoing sensor parameters.

The first line reported the name (acq_initialize) of the function, and the next line reported the sensor's initial (undegraded) range in meters. Then for each of the six OPFOR units in the vignette, a random number was drawn and echoed.* When the cumulative probability of acquisition of this unit by this sensor exceeded this random number, an acquisition would occur. Until then, the random number was simply stored.† Next, an acquisition check event was scheduled for each OPFOR platform. This was the event at which the cumulative probability of acquisition was updated. In this case the sensor was one that operated only in daylight, so the first acquisition check was not scheduled until 21600 s past midnight (that was 0600, the time at which night changed to day in the vignette) so that no effort was wasted on checking for acquisition with a sensor that cannot acquire in the dark. The parameters reported in each acquisition check event scheduled were the event execution time (so it could be placed in proper order on the event queue), the event type (et=3 indicated an acquisition check), the platform ID (pID=2 indicated platform no. 2, one of the ARVs), pID2 was not used, the OPFOR unit ID (rID=25 indicated the OPFOR unit that was mover number 25), the sensor range (f=3000 was the sensor range in meters), the sensor number on the platform (eq=2 indicated equipment 2, which was the third sensor on the platform since the C language numbering convention was used), and other parameters not used for this event type.

*If this is not the first sensor to be initialized on the platform, then the random number is weighted with the random number drawn for the first sensor in an attempt to account for dependence in what the sensors on the same platform acquire.

†If the unit entered and exited the sensor's coverage (as often happens with a moving UAV), then the random number would be reinitialized.

After the program had initialized the acquisition process for each sensor on each OWNFOR platform against each OPFOR unit, the program reserved space both at the C2V and at the alternate backup C2V for a fire mission against each OPFOR unit. This was echoed with a simple message ("Creating Fire Mission"), but nothing was filled in or scheduled at this time.

Next, the no-fire zones were read in and their vertices echoed. In this vignette, there was only one no-fire zone, and the information echoed is shown in figure H-10.

```
Now creating 1 no fire zones
    No fire zone 0 is described by this polygon:
 This polygon has 4 vertices as follow:
(91000,105000)
(93000,105000)
(93000,106500)
(91000,106500)
 minimum and maximum x values = 91000  93000
 minimum and maximum y values = 105000  106500
```

Figure H-10. SBM log file: echoing no-fire zone information.

This information display was self-explanatory, aside from reminding the reader that the C-language numbering convention was used, so that "No-fire zone 0" is really the first no-fire zone. X values were east-west coordinates in meters, whereas y values were north-south coordinates in meters.

Next, the program scheduled the unmanned aerial vehicle (UAV) recovery events at the preprogrammed times (figure H-11). There was one such event scheduled for each UAV played in the vignette.

```
event_schedule UAV recovery. tm=20520.000000 et=15 pID=3 pID2=0 rID=0 f=0.000000 eq=0
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule UAV recovery. tm=30780.000000 et=15 pID=4 pID2=0 rID=0 f=0.000000 eq=0
mt=0 non-message event ct=0 mptr=0     event_schedule exits
  event_schedule UAV recovery. tm=37800.000000 et=15 pID=5 pID2=0 rID=0 f=0.000000 eq=0
mt=0 non-message event ct=0 mptr=0     event_schedule exits
```

Figure H-11. SBM log file: scheduling UAV recoveries.

The relevant parameters in figure H-11 were the time (for example tm=20520 s after midnight) at which the event was to be executed, the event type (et=15 indicated UAV recovery), the platform ID number (pID =3 indicated the UAV with platform number 3); the other parameters were not used for this event type.

The program next read in and echoed the message routing table. That was a direct copy of the input described in appendix F, printed once with labeling and once without.

The situation awareness (SA) and common operating picture (COP) messages were scheduled next. For each OWNFOR platform, an event was scheduled at the beginning of the vignette to report its position to the C2V and its backup. Similarly, COP update messages were scheduled by the C2V and its backup 30 s after the vignette started.

The first external event (from the file of ECD state changes and task start-stop events) was scheduled to be read at the beginning of the vignette, and the first report of platform and unit moves was scheduled to be printed to the file that would drive the situational map display described in section H.2.

Before the vignette was executed, the program echoed the entire contents of the event queue. All of the initial movements, acquisition checks, UAV recoveries, SA messages, and COP updates that had been scheduled were printed out in a fairly terse format so that the developer could check them. An excerpt is shown in figure H-12.

```
    event 132 at time 7200.000000 and address 22472576 is type 4 with i, j= 22 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 42417672 prev=22472512 next=22472640
    event 133 at time 7200.000000 and address 22472640 is type 4 with i, j= 23 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 42418696 prev=22472576 next=22472704
    event 134 at time 7200.000000 and address 22472704 is type 4 with i, j= 24 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 42419720 prev=22472640 next=22472896
    event 135 at time 7200.000000 and address 22472896 is type 10 with i, j= 0 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 0 prev=22472704 next=22472960
    event 136 at time 7200.000000 and address 22472960 is type 13 with i, j= 0 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 0 prev=22472896 next=22472768
    event 137 at time 7230.000000 and address 22472768 is type 4 with i, j= 6 0 redt = 0  f =
0.000000  eq = 0 mt = 1  ct = 0 ptr = 19111944 prev=22472960 next=22472832
    event 138 at time 7230.000000 and address 22472832 is type 4 with i, j= 7 0 redt = 0  f =
0.000000  eq = 0 mt = 1  ct = 0 ptr = 19779592 prev=22472768 next=22121856
    event 139 at time 7500.000000 and address 22121856 is type 0 with i, j= 3 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 0 prev=22472832 next=22386224
    event 140 at time 10300.000000 and address 22386224 is type 0 with i, j= 17 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 0 prev=22121856 next=22398592
    event 141 at time 10336.000000 and address 22398592 is type 0 with i, j= 18 0 redt = 0  f =
0.000000  eq = 0 mt = 0  ct = 0 ptr = 0 prev=22386224 next=22410960
```

Figure H-12. SBM log file: excerpt of event queue at start of vignette.

In figure H-12, the event notices were read in as follows, taking the first one as an example (the first event in the text was the 132nd event on the queue):

- The vignette was scheduled to be executed at time 7200 (seconds after midnight).
- The event information was stored at memory location 22472576 (of interest only for debugging).
- Designated event type 4 (a "send message" event).
- Performed by platform 22 (one of the OWNFOR NLOS-Cs).
- Performed on neither OWNFOR unit (0) nor any OPFOR unit (redt = 0).
- The f and eq parameters were not used for this event type.
- The message type was an SA update (mt=0).
- The communications type was data (ct=0).
- The pointer to the address where message content was stored was 42417672 (again only of interest for debugging).

"Prev" and "next" were the addresses of the previous and next event notices in memory (again only of interest for debugging). This echoing of the initial event queue, like all of the previous data echoing, was mainly for the purpose of verifying that objects and events were correctly initialized.

Once initialized, the program was ready to begin the simulation by removing the first event from the event queue and executing. Each of the thousands of events executed and new events scheduled during the course of each replication of the vignette was echoed to the log file. It would be impractical to excerpt and explain each type and would probably only be of interest to a very small population of readers. However, to give a taste of what the printouts looked like, figure H-13 contains excerpts from the log file for the execution of three events. The amount printed out for each event execution varied greatly with the event type.

The first event was a message received at time 8134.532975 s after midnight by one of the NLOS-C (pID=18) from the C2V (pID2=6) telling it to cease firing because the fire mission had been completed. The second was a message received at time 8135.506245 s after midnight by the backup C2V ("ALT", pID=7) from the C2V copying it on the fact that the fire mission had been completed. The third event, however, was an external event previously read in from the script file and inserted in the event queue to occur at time 8141.2300; it printed out much more information.

```
{ event_execute 805 Receive Message. tm=8134.532975 et=5 pID=18 pID2=6 rID=27
f=0.000000 eq=0 mt=9 cease_fire ct=0 mptr=42856456
 Fire Mission completed !! condition 2.
} event_execute 805 exits
{ event_execute 806 Receive Message. tm=8135.506245 et=5 pID=7 pID2=6 rID=27 f=0.000000
eq=0 mt=13 fire_mission_done_copy ct=0 mptr=42857480
 ALT receives notice from C2V that fire mission against 27 is done
} event_execute 806 exits
{ event_execute 807 External Event. tm=8141.230000 et=10 pID=0 pID2=0 rID=0 f=0.000000
eq=0 mt=0 non-message event ct=0 mptr=0
Ext Event: time= 8141.2300 eev_type= 1 plat_nr= 0 plat_type= 1
  DCS:  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  Comp Kill counts:   0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
 CSV: 000000000000000000000000000000000200000000000000000000000000000000...
 Now dumping task pass/fail info for platform # 0
 Task Number 9 not currently required of this platform
 Task Number 11  M:fail  F:pass  A:pass  Z:pass  X:pass  S:pass  C:pass  P:pass  O:pass
K:pass overall p/f = fail
 ...
 Across Tasks p/f  M:fail  F:pass  A:pass  Z:pass  X:pass  S:pass  C:pass  P:pass  O:pass
K:pass
 overall p/f over tasks and categories = fail
  event_schedule Move speed change. tm=8141.230000 et=2 pID=0 pID2=0 rID=0 f=0.000000
eq=0 mt=0 non-message event ct=0 mptr=0    event_schedule exits
  event_schedule Send Message. tm=8141.230000 et=4 pID=0 pID2=6 rID=0 f=0.000000 eq=0
mt=2 DCS_status_update ct=0 mptr=0    event_schedule exits
...
 DCS_vector_print:  /M 0 0 0 1 /F - - - - 0 /A 0 0 /Z 0 0 0 - /X 0 - - - 0 /S - - 0 - - /C - - - - - - - /P *
/O - /K 0/
 health bar for platform 0 of type 1 for DCS category 0 Mobility
 number of bar tiers = 5
 Color of tier 0 is green
 Color of tier 1 is amber
 Color of tier 2 is red
 Color of tier 3 is red
 Color of tier 4 is red
 green-amber divide at 0.200000,  amber-red divide at 0.400000
 Mobility category. diamond color is red and position is 0.700000
...
 res realloc. pt, av_count, min_req = 1 2 0
 Current Health rating for platform 0 =  M 0.200 0.400 0 0.700 F 1.000 2.000 1 0.250 A 1.000
2.000 1 0.167 Z 0.500 0.750 1 0.125 X 0.500 0.500 1 0.250 S 1.000 2.000 1 0.250 C 1.000 2.000
1 0.500 O 1.000 2.000 1 0.500 K 0.500 0.500 1 0.250
 task # 11 is fail
 task # 16 is pass
 task # 23 is fail
```

Figure H-13. SBM log file: excerpt of event execution.

The third event in the excerpt was an externally scripted component failure on the first ARV. The printout reported the ECD state (the fourth degradation element, M4, has been triggered), the number of components relevant to each degradation element (one), and the component state vector (by converting the hexadecimal to binary and counting to the position of the component in this bit string, one can obtain the actual number of the component that failed). Next, the printout looped through the tasks that a platform of this type might be required to perform, and for each

such task currently required of it, it looped through the degradation categories (mobility, firepower, etc.) and assigned a pass/fail rating depending on whether the platform had enough capability in that category to perform the task. Then it generated an aggregate pass/fail rating that is the "worst case" for the tasks currently required of the platform.

In this example, the platform suffered a total immobilization, so a speed change (to zero) is scheduled for immediate execution, and a message to the C2V to report its change in status is also scheduled for immediate sending.

The updated ECD vector was reported and now shows a "1" in the M4 position.

Next, the program generated the health bar status for the platform considering its new state and its current workload of required tasks. To do this, the program looped through the ECD categories, and for each task currently required of the platform, it generated a color rating for that category and task set.* The color of each tier for the current category was then reported, and the position and color of the diamond was calculated.† This information was summarized in a "current health rating" for the platform. A letter was printed for each of the 10 ECD categories (M for mobility, F for firepower, etc.) and four numbers that respectively indicated where (on a scale from 0.0 to 1.0) the health bar should change from green to amber, where it should change from amber to red, whether the diamond should be colored red (0) or green (1), and where (on the scale from 0.0 to 1.0) the diamond should be located.‡ Note that in the final version of the health bars, as described in section H.3 the order of the colors in the bars was changed from green-amber-red to red-amber-green, but the meaning of the colors was unchanged.

The program then printed lines (three in this example) citing the tasks that the platform was currently required to perform and the pass/fail status of each. Then it read in the next external event from the script file and added to the event queue in time sequence (to be executed at time 8160.603 s after midnight in this example). That completed the execution of the event.

The program continued in this fashion, logging the execution of all events during each replication until it encountered the first event with an execution time after the vignette end (36000 s after midnight or 1000).§ When the program reached the end of the replication, it printed out a number

*The colors were initialized for each task and degradation tier, as described for figure H-13. At this point, the "worst" color for each tier within each category was compared to the set of tasks currently required of the platform. Red was worse than amber, and amber was worse than green. Red indicated that all states at the given tier of the category failed the current set of required tasks, amber indicated that some states at the given tier of the category failed some of the required tasks, and green indicated that all states at the given tier passed all of the currently required tasks.

†The diamond color indicated whether the platform in its current state had sufficient capability to pass all of its required tasks. If so, it was colored green; if not, it was colored red. The position of the diamond was the center of the tier in which its damage state falls.

‡A value of 2.0 for the amber to red color change indicated that it never changed to red (usually because no capability was required in that category to perform the required tasks).

§There was always an end-of-replication event put on the event queue with a time 1 min after the intended vignette end. In addition, the simulation would end the replication any time it encountered a null pointer to the next event.

of things of possible interest to the developer, debugger, or analyst. First it printed out the end state of each OWNFOR platform and OPFOR unit showing its state (often not the same as the pristine state printed out at the initialization of the vignette). Since this printout used the same format as the echoing of the initial conditions, it was not repeated here.

There was a summary intended mainly to show that acquisitions were happening by indicating which sensors acquired each unit. Figure H-14 shows an example of the C2V and the ALT (code name for the backup C2V) perception of the OPFOR status.

```
C2V and ALT perceptions
 C2V's perception at end of replication
   Red unit 0 25 red_OP at (89000,102500) perceived at (89000,102500) .  Fire Mission completed
         includes dead apc at (87779.1,103597)
   Red unit 1 26 red_ATC at (91500,105300) perceived at (91500,105300) .  In No Fire zone
   Red unit 2 27 red_HQ at (90500,102500) perceived at (90500,102500) .  Fire Mission completed
   Red unit 3 28 red_P_A at (89000,102500) perceived at (89000,102500) .  Fire Mission completed
         includes dead apc at (89068.7,102450)
         includes dead apc at (89097,102491)
   Red unit 4 29 red_P_B at (91700,102300) perceived at (91700,102300) .  Fire Mission completed
         includes dead apc at (91611.8,102333)
   Red unit 5 30 red_FA at (91500,100500) perceived at (91500,100500) .  Fire Mission completed
         includes dead sph at (91570.1,100516)
         includes dead sph at (91442.9,100481)
 ALT's perception at end of replication
   Red unit 0 25 red_OP at (89000,102500) perceived at (89000,102500) .  Fire Mission completed
         includes dead apc at (87779.1,103597)
```

Figure H-14. SBM log file: end of replication perception of situation.

This simply reported for each red (OPFOR) unit its number (among red units), its number among mover objects in the simulation, its actual coordinates at the end of the vignette, and its perceived coordinates at the end of the vignette. There was also a string indicating whether a fire mission was completed against it or whether it was in a no-fire zone. In the case of those units where some platforms were killed by OWNFOR fire missions, the killed platforms were listed by type and coordinates. Then this information was repeated for the backup C2V (ALT). Unless communications were cut between the C2V and the ALT, this would be the identical printout.

Although this demonstration focused on the ability to complete the tasks constituting the mission, the model still printed out the results of the fire missions against the OPFOR. An example from one replication is included in figure H-15.

```
Smart munition results:
 red lost 4  apc
 red lost 2  sph
 number of hits that didn't kill = 2
 number that hit false targets = 1
 number that hit dead hulks = 2
 number that missed everything = 13
 total number fired (subm) = 24
 mean distance (mils) from submunition to tgt center = 7.899376
 mean distance (mils) from dispense point to target center = 7.104958
```

Figure H-15. SBM log file: summary of smart munition effects.

The printout was a summary of smart munition results in the vignette. In this replication, 24 smart submunitions were fired. They killed four armored personnel carriers and two self-propelled howitzers; 2 of them hit targets without a kill, 1 hit a false target, 2 hit dead hulks (already killed targets), and 13 missed everything (live targets, false targets, and dead hulks). Finally, the mean distance (in mils) from submunition to target center was reported as was the mean distance in mils from the carrier dispense point to the target center.* These last two lines were printed as a check that the delivery error sampling was producing reasonable results.†

The next step in wrapping up the replication was to print out the remaining unexecuted events on the event queue. This was solely for debugging purposes and resembles the format in a previous example (figure H-13).

One more interesting printout reported whether the commander's intent was achieved on the replication; this report is shown in figure H-16.

```
Final OWNFOR status:
2 fully capable arv at end of replication and 1 not fully capable arv
3 fully capable uav at end of replication and 0 not fully capable uav
2 fully capable c2v at end of replication and 0 not fully capable c2v
9 fully capable mcs at end of replication and 0 not fully capable mcs
5 fully capable nlos_c at end of replication and 1 not fully capable nlos_c
 On replication 3 Commander's intent achieved
```

Figure H-16. SBM log file: commander's intent.

*In this vignette a notional munition containing two smart submunitions was used by the NLOS-C. Each round arrived at its burst point and dispensed the two submunitions randomly over a 100-m-radius circle. The submunitions then searched the ground below for targets.

†If one shooter suffered a degraded delivery accuracy state, it would show up with an atypically large mean miss distance; however, printouts of the actual fire mission execution events allowed that to be checked directly rather than by inference.

This printout simply reported the number of fully capable platforms in the OWNFOR at the end of the replication. The specified "Commander's Intent" was achieved if at least one of the C2Vs (or its backup), seven of the Mounted Combat Systems (MCSs), and four of the NLOS-Cs were functional at the end of the vignette and had reached their objective in the case of the MCS and C2V (or backup). On this replication, the commander's intent was achieved.

The remainder of the log file for each replication consisted of some messages that various objects were deleted. This was merely for debugging to make sure the memory management was occurring correctly. If more replications were to be performed, the program started over with the initialization of the next replication as described previously.*

H.2 Event History File and Situation Map

The SBM output an event history file used to drive a map display program. The map display program was implemented as a postprocessor to the SBM. It began with a display of the initial positions of the OWNFOR and OPFOR. The postprocessor then moved the platforms and units around the map as the vignette was played back. It showed sensors' fields of view, message traffic, weapon firing, and movement of platforms. The postprocessor display was a simple graphical view of the vignette as it executed where the effect of degradations could be seen. For example, one could watch immobilized platforms come to a stop, degraded sensors having reduced fields of view (or none at all), degraded weapons that do not fire, degraded communicators that do not send messages, and catastrophically killed platforms ceasing to function altogether.

The screenshot in figure H-17 shows the map at time 7806.73 (0210 and 6.73 s). The various circular sectors (in orange, violet, and green) represent sensor coverage. The red dots were the OPFOR units (labeled, for example, "Enemy Unit 29"); the two red dots circled in black were OPFOR units that had been fired on at this point in the vignette. The blue dots were the OWNFOR platforms (appropriately labeled). Since the figure was derived from a screenshot and scaled to fit the paper page, it is not as sharp as the original on a good monitor, but one can get a sense of the information displayed. The black line segments connecting some of the OWNFOR platforms represented message traffic. There were no actual weapon firings in progress in this screenshot (though the circled OPFOR units indicated some have taken place), and it was impossible to see the movement of the platforms about the terrain in a static screenshot. In an actual dynamic playback, it is possible to see movements, weapons firings, frequent message traffic, etc.

*The only changes being that the random number stream was at a different point and that a different script of combat damage, failures, and repairs would be used for the next replication (the script of start and stop times for the tasks remained the same from replication to replication).

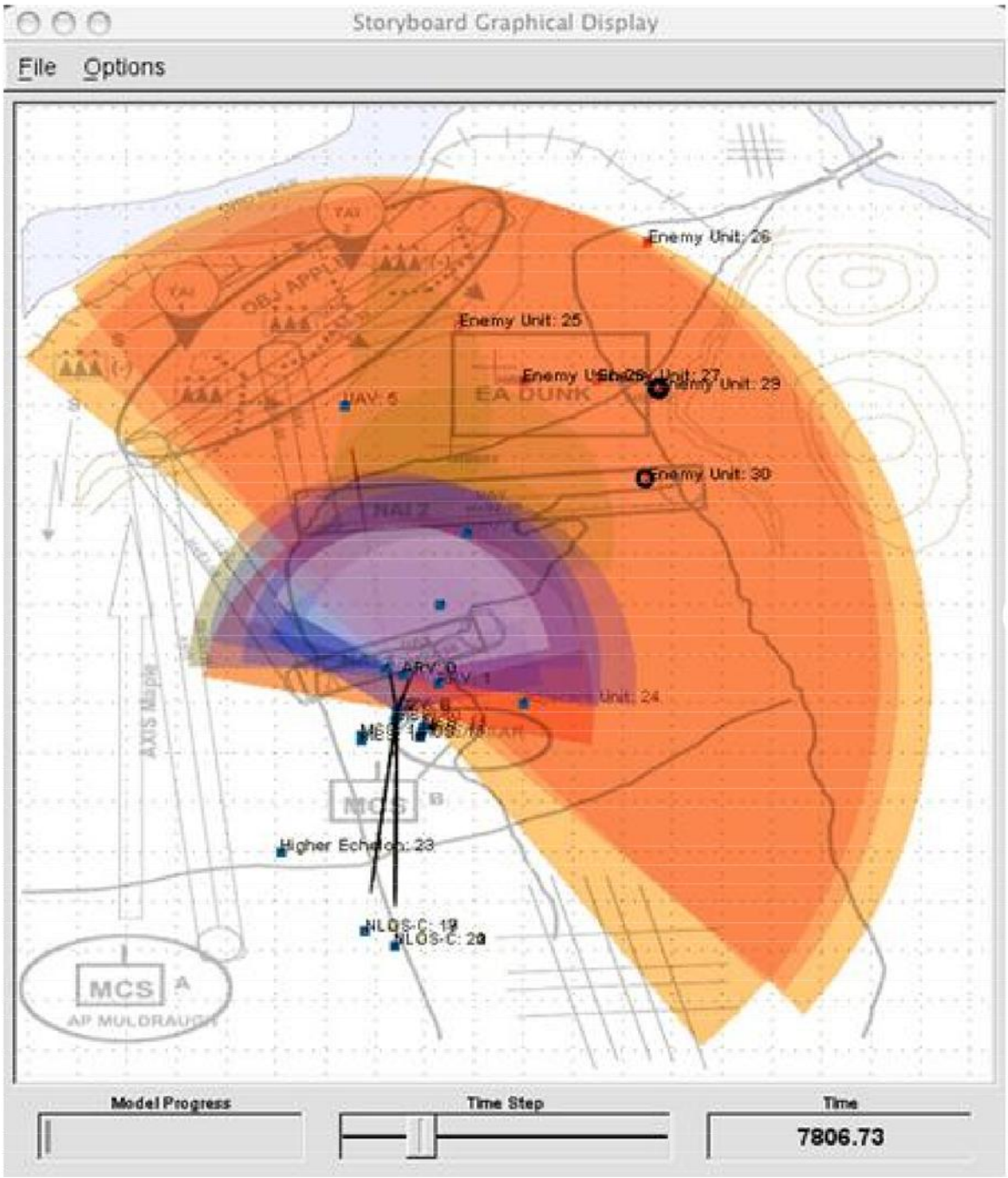


Figure H-17. Screenshot of map playback.

To drive the map display, the SBM wrote an event history file saving information, such as initial conditions and state changes of the platforms and units during a replication of the vignette. This file consisted of one-line printouts of relevant parameters as each platform and unit was initialized and as each event was executed. The printouts were the minimum information needed for the map display. Figure H-18 shows a small sampling of typical lines in the event history file.

```
tm=7920.7048 et=5 sndrID=3 x=88252.27 y=100948.38 rcID=7 x=863000 y=950000 mt=0 ct=0
tm=7933.2250 et=7 pID=17 x=930000 y=855000 tx=89059.30 ty=102467.54
tm=7934.2478 et=7 pID=18 x=930000 y=855000 tx=89096.51 ty=102515.64
tm=7937.1190 et=10 pID=21 eet=2
tm=7939.4790 et=3 pID=3 x=88154.73 y=100638.58 sr=0 rg=2996.13 an=3600 orx=0.2631 ory=0.9647
tm=7950.4035 et=5 sndrID=3 x=88102.12 y=100471.46 rcID=6 x=863000 y=950000 mt=0 ct=0
tm=7985.0224 et=8 shID=17 x=930000 y=855000 tx=89059.30 ty=102467.54
tm=7985.4633 et=5 sndrID=17 x=930000 y=855000 rcID=7 x=863000 y=950000 mt=6 ct=0
tm=7988.0712 et=0 pID=25 x=877000 y=1036000 vx=4.241 vy=-3.588
tm=7988.0712 et=9 pID=2 x=862000 y=951000 sr=0 rg=100000 an=1800 orx=0.1737 ory=0.9847
tm=8141.2300 et=2 pID=0 x=863000 y=950000 vx=00 vy=00
tm=8141.2300 et=3 pID=0 x=863000 y=950000 sr=0 rg=100000 an=1800 orx=0.1692 ory=0.9855
tm=8250.1244 et=1 pID=3 x=866000 y=957000 vx=00 vy=00
tm=8250.1244 et=0 pID=3 x=866000 y=957000 vx=1.695 vy=16.580
```

Figure H-18. Event history file excerpt.

The first line shows that the event was executed at time 7920.7048 s after midnight and that it is an event type 5 (receipt of a message). The sender of the message was platform 3 (one of the UAVs), and the sender's coordinates were (88252.27, 100948.38). The receiver was platform 7 (the backup for the C2V), and the receiver's coordinates were (863000,950000). The message type was 0 (an SA update), and the communications mode was 0 (data). This was all of the data the map display needed to draw a line between the two platforms indicating message traffic (the line would persist on the display only long enough to be visible to a human observer).

The other sample lines were similar except that the exact parameters differed from one event type to another. The second line, for example, was a firing event; it provided the time, event type (7), number (17) and coordinates (x,y) of the shooter, and coordinates of the aim point (tx,ty). This was followed later by an event type 8 (rounds arrive), which had the same parameters. This information allowed the display to draw a nominal trajectory from the firing platform to the aim point (and erase it when the round arrived). Other event types in this excerpt included an event type 3 (acquisition check), which provided a two-dimensional unit vector (orx,ory) in the direction the sensor was pointing; an event type 9 (sensor reorientation), which provided the same parameters; and an event type 0 (departure from a node), which included not only the current position (x,y) of the platform but also its velocity vector (vx,vy). Event types 1 (arrival at node) and 2 (speed change) included the same parameters as event type 0.

Event type 10 was somewhat different in that it indicated an external event (one from the script) was read in at the indicated time. In the example it was an external event type 2 (eet) as applied to platform 21 (one of the NLOS-C). Event type 2 was a component repair event. The information on which component was repaired was not written to this file, as components are not displayed on the map; however, if the repair restored a capability to move, shoot, communicate, or acquire to the platform, then subsequent events using those capabilities would result in actions visible on the map display.

H.3 Health History File and Health Meter Display

Another graphical postprocessor allowed one to watch the health of the platforms and of the entire OWNFOR change as the vignette executed. This display was also implemented as a postprocessor driven by a health history file. The history file provided an instantaneous* comparison of each OWNFOR platform's capability against the requirements of the tasks currently demanded of it. There was then an aggregate (or collective) OWNFOR health assessment showing the instantaneous capability of the force to perform the tasks required of its mission.

Figure H-19 shows the display for critical mission tasks, and figure H-20 shows the process to determine the collective task health of the unit and mission. There are four steps and three rules in the process of determining the collective task health of the unit and health of the mission. Enter the "collective task health meter" at step 1 and follow through to step 4 for each collective task:

1. If any bar is red or amber, go to step 2.
2. If the acceptable risk is "no," go to step 3.
3. If the adjustments bar is red, go to step 4.
4. If the acceptable risk is "no," the task bar (shown in figure H-20) was red.

The rules were as follows:

1. When any platform task is degraded, the associated collective task turns amber.
2. If a critical platform task turns red while it is being performed, the associated collective task turns red.
3. If the collective task bar is red and associated risk is "no," then step 3 may be repeated and MoPs reviewed again until step 4 is a "yes," or until there is no more time, or until there are no more resources.

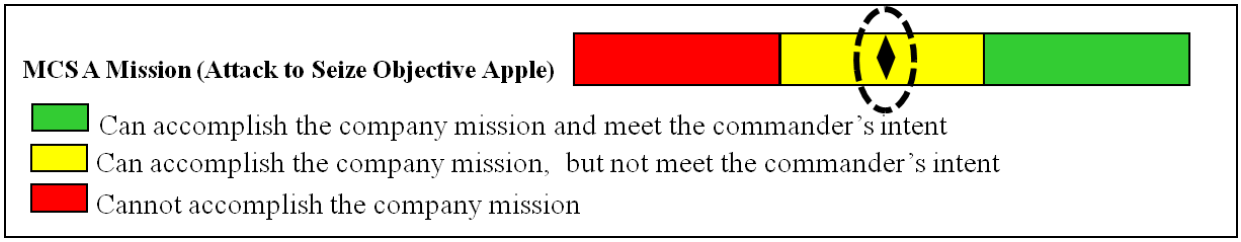


Figure H-19. Tasks that impact mission success.

*_Instantaneous_ means that the display shows whether the capabilities at each instant were sufficient to perform the tasks required of the platform at that instant. This is not the same as showing whether the platform successfully completed the task. The platform may have been required by the vignette script to be capable of maintaining surveillance of an area for an hour, but even if it were incapable for some of that time, it may still have been capable long enough to acquire and call for fire against the OPFOR units in that area. Therefore, instantaneous capability does not correlate perfectly with task success or failure.

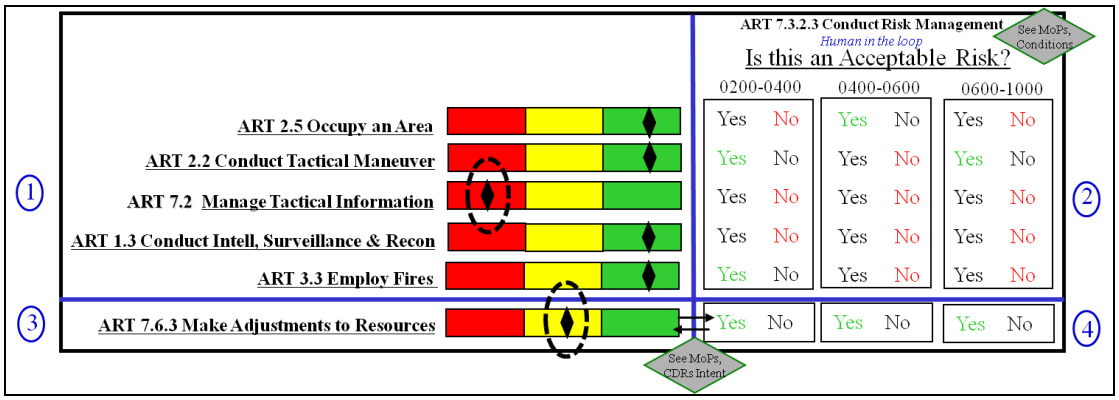


Figure H-20. Collective task health meter.

Note that step 2 was time sensitive as to when degraded tasks were an acceptable risk. Also note that measures of performance (MOPs) and conditions were reviewed in step 2, and MOPs and commander's intent were reviewed between steps 3 and 4 to help determine an accurate yes–no answer. If step 4 was a "no," step 3 may be repeated, then MOPs reviewed again until step 4 was a "yes" or until there was no more time, or until there were no more resources. The "mission health" would change based on the results of steps 1–4. Collective task health was linked to mission health but would not severely impact overall mission health unless the degraded collective tasks were so critical that a loss of the ability to perform those collective tasks resulted in an unacceptable risk that could not be rectified.

Figure H-21 is a screenshot of the health meter and health bar display for the simulation. On the left side of the figure is a row for each ECD capability (mobility, firepower, etc.) and a column for each OWNFOR platform whose capabilities were being tracked. A green circle indicates that the platform represented in that column had sufficient capability in the category corresponding to that row to perform its current set of required tasks. A red circle (UAV number 1 had several) indicates that the platform did not have sufficient capability in the category to perform its current set of missions; in the screenshot, UAV number 1 had insufficient capability in mobility, surveillance, and communications to perform its current required tasks. The red circle in the "Catastrophic" row indicated that UAV 1 had suffered a catastrophic kill and therefore had no capability to perform any tasks. This display was updated every time there was a change in platform capability state or task workload.*

The user could click on an individual platform's column and obtain the more detailed information in the two windows at the right of the screenshot. The top one of the two windows displays the health bars. For each of the nine ECD categories† there was a bar colored with red, yellow, and green segments (or some subset of those three colors) and a colored diamond symbol. The possible states of the platform with respect to each capability were grouped into tiers from

*In the case of UAV 3, the circles were empty because the UAV had not yet entered the vignette at that time.

†The ECD category "Passengers" was not shown because none of the OWNFOR platforms in this vignette carried passengers; they were either unmanned or carried only crew personnel.

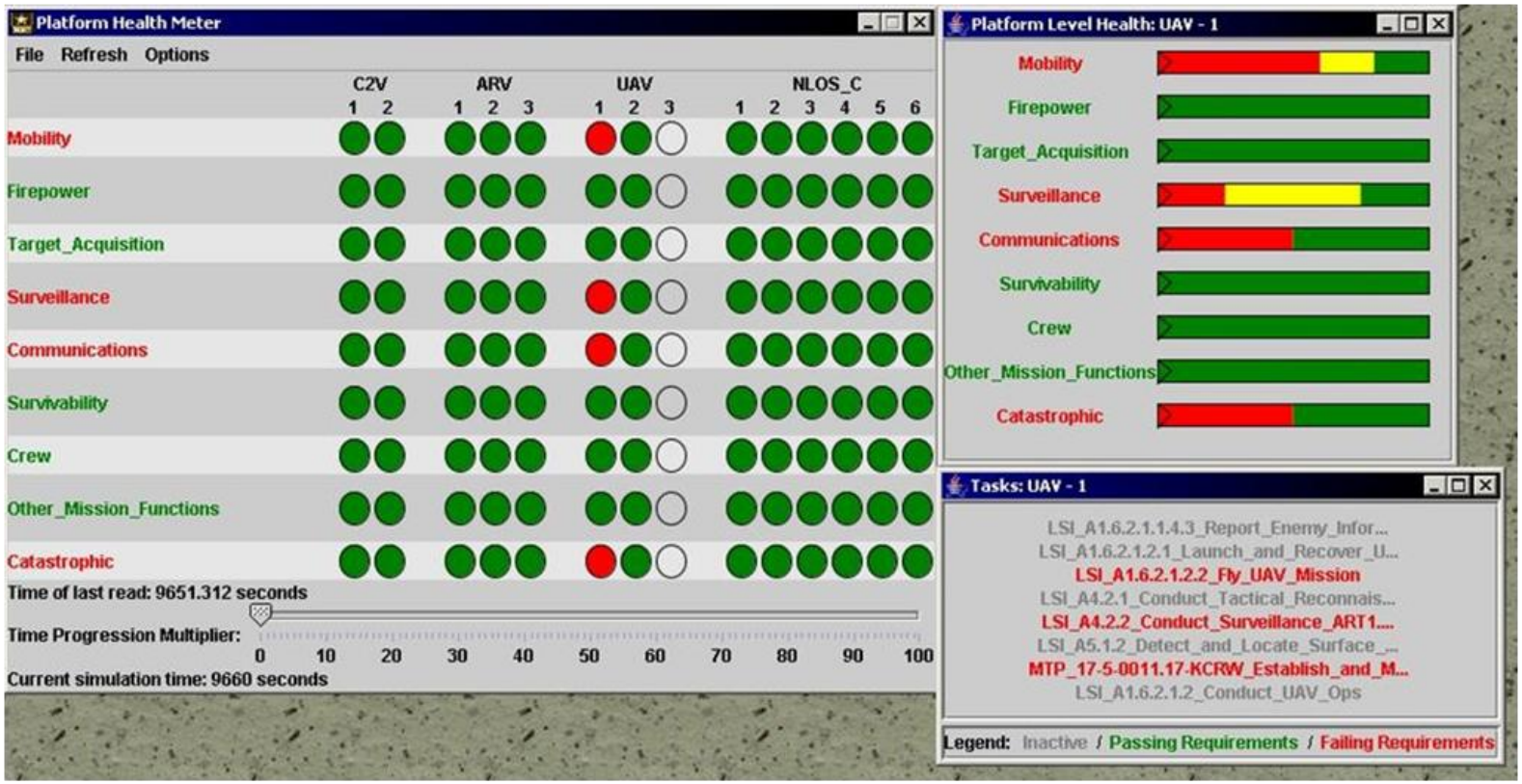


Figure H-21. Health meter display.

undamaged to loss of all capability in that category.* For example, in the mobility category, there were five tiers defined as follows:

- Tier 0 is undamaged.
- Tier 1 means the platform had suffered one of the degradation elements {M1, M2, M3}.
- Tier 2 meant it had suffered two of the degradations.
- Tier 3 meant it had suffered three of the degradations {M1, M2, M3, M4}.
- Tier 4 meant it had suffered all four of the degradation elements {M1, M2, M3, M4}.†

For each tier, an evaluation was made for the states in that tier to determine whether a platform in each such state could perform all of the tasks currently required of it. If it had the capability to perform the task given any of the states at that tier, then that tier was assigned the color green. If every state at that tier yielded a platform with capabilities inadequate to perform its current task workload, then the tier was assigned the color red. Finally, if for some states in that tier the platform could perform all of its currently required tasks but for other states it could not, then the tier was assigned the color yellow.

The display bar for that category was then divided into segments corresponding to the tiers and colored accordingly. In the case of mobility, the bar was divided into five equal segments running (left to right) from tier 4 to tier 0 and colored according to the scheme described previously. In this case it looks like a platform at any state in tiers 4, 3, or 2 would fail all of the current task workload demanded of this platform. Depending on which state at tier 1 the platform was in, it would either be able to perform all of the tasks in its current workload or not (as indicated by the yellow color), and a platform in tier 0 would be able to perform all of its current task workload (it is in the undamaged state, after all).

The health bar provided an appreciation of how difficult the current task workload was on the platform. The more green, the easier the workload of tasks; this implied a tolerance for more degradation before any of the currently required tasks would be beyond the capability of the platform. More red in the health bar indicated a lower tolerance to degradation before causing failure at one or more required tasks. In the case of a task workload that required no capability in a given category (such as firepower in the example of UAV 1 in the screenshot), the bar would be entirely green (all tasks in the current workload could be performed even with no capability in that category).

*This is explained in greater detail in appendix I.

†Because degradation element M4 (total immobilization) implied that degradation elements M1 (reduced maximum speed) and M2 (reduced maneuverability) had occurred, some combinations (such as M1 and M4 without M2) could not occur.

The health bar was a function of the ECD category, the platform type, and the currently required task workload, but it did not show anything about the current platform state. To show that, a diamond was drawn on the bar to indicate which tier it currently fell into. The diamond was filled in with red or green, depending on whether the current platform capabilities in that ECD category were sufficient to perform the current task workload. Note that a diamond would never be colored yellow. Yellow means that there was more than one possible outcome at a given tier because that tier included multiple combinations of degradation elements; however, a single platform would never simultaneously be in multiple states, so it was either red or green. In the example of UAV 1 shown in figure H-22, all of the diamonds were at the leftmost point of their respective bars because the UAV 1 had suffered a catastrophic kill and was in the most severe damage tier in every category. In general, the diamond would move back and forth along the bar as the platform state and task workload changed.

Finally, in the lower-right corner of figure H-21 is a text listing of the tasks for the selected platform type (UAV in this case). All tasks that the platform type might be required to perform at some point in the vignette are listed with a brief alphanumeric description (along with the lead system integrator [LSI] or other task list number). Those tasks not currently required of the platform were printed in gray, those that the platform was currently required to perform but could not were in red, and any that it was required to perform and was able to would be printed in green.

The larger left window also displayed the time the display was last updated (that was at 9651.312 s after midnight or 0240 and 51.312 s), and a control bar at the bottom of the window allowed the user to control the playback speed so that the changes in status did not just flash by in an instant.

The health meter/health bar display could be executed in synch with the previously described map display. This allowed one to watch the platforms move about the map and to observe the changes in their health. For example, if a platform came to a stop, one could look at the health information for that platform to see whether it lost mobility or was still mobile but just reached a point where it was scripted to stop in the vignette.

The display program, written in JAVA, required five text input files to run: the vehicles file, the categories file, the task history file, the task index file, and the health history file. The first three files were used for the platform-level health displayed in figure H-22.

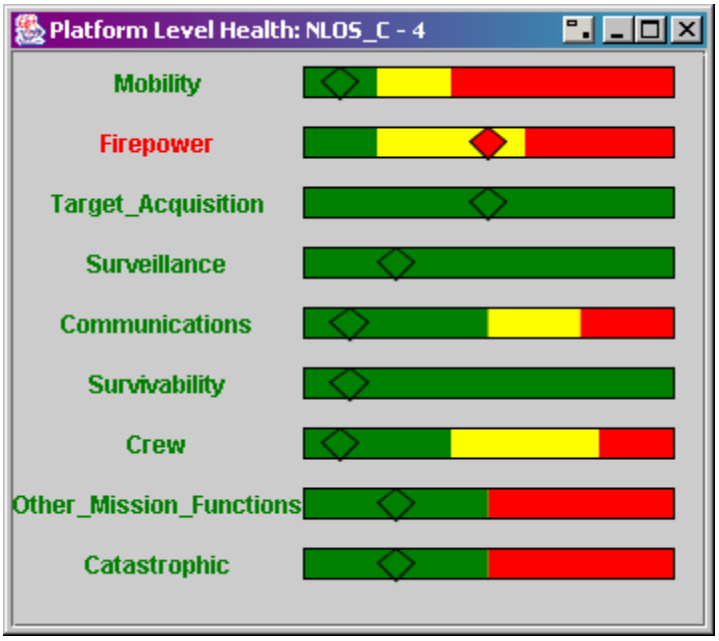


Figure H-22. Platform-level health display.

The vehicles file is a basic listing of the vehicle name and number, as shown in figure H-23. For the four vehicle types (C2V, ARV, UAV, and NLOS_C), the individual platforms were identified in this file. For example, on the first line there were two C2Vs labeled 1 and 2. Similarly, on the second line, there were three ARVs labeled 1, 2, and 3. This convention was continued for all four vehicle types.

```
C2V   1   2
ARV   1   2   3
UAV   1   2   3
NLOS C    1   2   3   4   5   6
```

Figure H-23. Vehicles file.

The categories file is a listing of the ECD elements. The main driver of the health display was the health history file generated by the SBM. An excerpt is found in figure H-24.

8141.230 arv 0 Mobility 0.600 0.800 0 0.300 Firepower 0.000 0.000 1 0.750 Target_Acquisition 0.000 0.000 1 0.833 Surveillance 0.250 0.500 1 0.875 Communications 0.500 0.500 1 0.750 Survivability 0.000 0.000 1 0.750 Crew 0.000 0.000 1 0.500 Other_Mission_Functions 0.000 0.000 1 0.500 Catastrophic 0.500 0.500 1 0.750

Unit-level task info w/o risk assessment
Task group ART 2.5 Occupy an area is fail color red
Task group ART 2.2 Conduct tactical maneuver is pass color green
Task group ART 7.2 Manage tactical information is pass color green
Task group ART 1.3 Conduct ISR is fail color red
Task group ART 3.3 Employ fires is pass color green

Unit-level task info with risk assessment
Task group ART 2.5 Occupy an area is fail color red
Task group ART 2.2 Conduct tactical maneuver is pass color green
Task group ART 7.2 Manage tactical information is pass color green
Task group ART 1.3 Conduct ISR is fail color red
Task group ART 3.3 Employ fires is pass color green

Mission-level pass/fail without reallocation: fail
Mission-level pass/fail after resource reallocation: pass

8160.603 c2v 6 Mobility 0.600 0.800 1 0.900 Firepower 0.000 0.000 1 0.500 Target_Acquisition 0.000 0.000 1 0.833 Surveillance 0.750 0.750 1 0.875 Communications 0.400 0.800 1 0.900 Survivability 0.000 0.000 1 0.900 Crew 0.333 0.667 1 0.917 Other_Mission_Functions 0.500 0.500 1 0.750 Catastrophic 0.500 0.500 1 0.750

Figure H-24. Health history file excerpt.

The health history file was printed out by the SBM but filtered (slightly) before use by the health meter postprocessor. The information consisted of an entry for each time there was either a change in some platform's capability state (because of combat damage, component failure, or repair) or a change in some platform's workload of required tasks (new task required or old task no longer required). The first four lines in figure H-24 show an example of the printed information at the time of state change. The first number is a time stamp (in seconds after midnight), followed by a platform type string (arv in this case), and a platform index number (0 indicating the first platform in this case). What follows is the information needed to construct the health bar display and the diamonds. This information consists of five fields for each of the ECD categories:

1. The category name as an alphanumeric string.
2. The point on the bar where the color should change from red to yellow.
3. The point on the bar where the color should change from yellow to green.

4. A Boolean flag to indicate whether the platform has sufficient capability to perform its task workload and hence what color it should be (1 indicates it has the capability and should be colored green, 0 indicates that there was at least one required task that it could not perform and should be colored red).

5. The position on the bar where the diamond should be located.

The positions were given on a scale of 0.0 to 1.0 and were scaled to the actual size of the bar in the postprocessor program.* The circles for the platform for each category in the left window of the display were also colored using the same color coding as for the diamond.

The program then printed out some information that was not needed by the postprocessor, which consisted of a summary of whether the capability at the unit level was still sufficient to complete the mission. This was simply the result of re-evaluating the process flow chart in figure H-20 to determine whether any of the currently critical collective tasks failed. The first evaluation assumed that all collective tasks were critical (no risk assessment), and then the remaining passes through the decision process took into account those tasks currently deemed critical (which changed with the phase of the vignette). Finally, two lines were printed that indicated whether the OWNFOR currently had capabilities to successfully complete the mission without resource reallocation and then whether it had capabilities with resource reallocation. In this example, several collective tasks were failing (or more precisely, the OWNFOR had insufficient instantaneous capability to perform them), and they were (according to the risk assessment) currently critical tasks. If no resource reallocation was performed and the current capabilities and critical tasks remained unchanged, the mission was predicted to fail. If a reallocation of resources was made, it was still assessed as possible to complete the mission successfully.

Then the SBM printed out the bar and diamond information resulting from the next change in platform state or task load, which occurred at time 8160.603 s after midnight. The postprocessor would then update the health display for this vignette time. The SBM continued printing out the collective task and mission success/failure assessments as before and printing out information for the next platform state or task load change, and so on until reaching the end of the vignette.

There was a small reformatting step before this health history file was passed to the postprocessor that merely renumbered the OWNFOR platforms. Instead of counting the platforms from 0 to 22 without respect to type, it numbered from 1 to N within each type (e.g., the three UAVs, which were numbered platforms 3, 4, and 5 in the SBM, were renumbered to be UAV 1, UAV 2, and UAV 3 since that was what the postprocessor expected). The reformatting also stripped out collective (unit-level tasks) tasks and mission success/failure assessment because the postprocessor did not use that information.

*A value of 0.0 for field 2 indicates that the bar for that category contains no red (and consequently starts with yellow or green at the left endpoint. A value of 0.0 for field 3 indicates that the bar for that category contains neither red nor yellow. Similarly, a value of 1.0 for either field indicates the absence of both green and yellow or of green alone on the bar.

The task index file (shown in table H-2) for the display program was a list of the tasks that could be performed by each vehicle. The first element on a line was the task index. This index was used to synchronize to the task history file. The second element was the task identification number but was not used by the display program. The third element was the task description, which would be displayed in the associated "tasks" window, as previously shown in figure H-21. The fourth element was the platform type assigned to the task.

Table H-2. Task index file.

0	ART3.3	ART3.3.1.1_Conduct_Surface_to_Surface_Attack	NLOS_C
1	ART7.2	ART7.2.5_Disseminate_Common_Operational_Picture_and_Execution_Information	C2V
2	ART2.2	LSI_A1.2_Conduct_Tactical_Maneuver_ART2.2	C2V
3	ART2.2	LSI_A1.2_Conduct_Tactical_Maneuver_ART2.2	NLOS_C
4	ART7.2	LSI_A1.6.2.1.1.4.3_Report_Enemy_Information	ARV
5	ART7.2	LSI_A1.6.2.1.1.4.3_Report_Enemy_Information	C2V
6	ART7.2	LSI_A1.6.2.1.1.4.3_Report_Enemy_Information	UAV
7	ART1.3	LSI_A1.6.2.1.2.1_Launch_and_Recover_UAV	UAV
8	ART1.3	LSI_A1.6.2.1.2.2_Fly_UAV_Mission	UAV
9	ART1.3	LSI_A4.2.1_Conduct_Tactical_Reconnaissance_ART1.3.3	ARV
10	ART1.3	LSI_A4.2.1_Conduct_Tactical_Reconnaissance_ART1.3.3	UAV
11	ART1.3	LSI_A4.2.2_Conduct_Surveillance_ART1.3.4	ARV
12	ART1.3	LSI_A4.2.2_Conduct_Surveillance_ART1.3.4	UAV
13	ART1.3	LSI_A5.1.2_Detect_and_Locate_Surface_Targets_ART3.2	ARV
14	ART1.3	LSI_A5.1.2_Detect_and_Locate_Surface_Targets_ART3.2	UAV
15	ART7.2	MTP_07-1-1COP.07-C332_Establish_the_Common_Operational_Picture	C2V
16	ART7.2	MTP_17-5-0011.17-KCRW_Establish_and_Maintain_Communications	ARV
17	ART7.2	MTP_17-5-0011.17-KCRW_Establish_and_Maintain_Communications	C2V
18	ART7.2	MTP_17-5-0011.17-KCRW_Establish_and_Maintain_Communications	UAV
19	ART2.2	LSI_A1.2.1.1_Employ_Traveling_Movement_Technique_ART2.2.1.1	C2V
20	ART2.2	LSI_A1.2.3.3_Exploit_Terrain_to_Expedite_Tactical_Movements_ART2.2.5	C2V
21	ART2.2	LSI_A1.2.4.7.3_Negotiate_a_Tactical_Area_of_Operations_ART2.2.12	ARV
22	ART2.2	LSI_A1.2.4.7.3_Negotiate_a_Tactical_Area_of_Operations_ART2.2.12	C2V
23	ART2.5	LSI_A1.5.2_Occupy_an_Attack/Assault_Position_ART2.5.2	ARV
24	ART2.5	LSI_A1.5.2_Occupy_an_Attack/Assault_Position_ART2.5.2	C2V
25	ART2.5	LSI_A1.5.2_Occupy_an_Attack/Assault_Position_ART2.5.2	NLOS_C
26	ART1.3	LSI_A1.6.2.1.2_Conduct_UAV_Ops	UAV
27	ART7.2	LSI_A2.3.1_Collect_Relevant_Information_ART7.2.1	C2V
28	ART3.3	MTP_06-5-A008_Conduct_Fire_Missions	NLOS_C
29	ART3.3	MTP_07-1-3000.07-C332_Employ_Fire_Support	C2V
30	ART7.2	MTP_07-1-WT06.07-C332_Conduct_Battle_Tracking	C2V

H.4 Task History File

The SBM also printed out a task history file as it executed. This file was used by both the health meter display postprocessor and the human analyst. An excerpt is shown in figure H-25.

```
time= 7899.651000 nlos_c 21 28 0 1
  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0 10  2  2  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
00000000000000000000000000000000000000000000000000000000000000000000000000000146a82c0000000000000000
00000000000000000000000000000
time= 8141.230000 arv 0 4 0 1
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
00000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
time= 8141.230000 arv 0 9 0 1
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
00000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
time= 8141.230000 arv 0 11 1 0
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  1  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure H-25. Task history file excerpt.

This file consists of multiline records, each beginning with the string "time=". What follows is the current time in the vignette for the information being printed (in seconds after midnight), an alphanumeric string of the platform type, the integer index number of the platform, a task number, a Boolean indicating whether the task was currently required of the platform, and another Boolean indicating whether the platform currently had the capability to perform the task.

The second line of each record is a line of Boolean variables indicating which ECD elements were in effect based on simple fault tree evaluation. The third line is the same except that implications were taken into account. For example, an M4 degradation element implied both M1 and M2; hence, in any of the excerpted records for the ARV, a "1" in the fourth field of the third line was always accompanied by a "1" in each of the first two fields. The fourth line is a simple count of the number of components killed on the fault tree for the corresponding element of degradation on the second line. Finally, the fifth line of each record is the component state vector in hexadecimal. By converting it to binary and counting the positions of any 1s that may have occurred, we found the actual indices of any components that were dysfunctional at that time in the vignette.

In the case of the ARV, we could see that it had suffered an M4 degradation (fourth field of the second line corresponds to M4), which implied both M1 and M2 degradations (first and second fields of third line). Also, there was exactly one dysfunctional component on the M4 fault tree, and the component index number is 126 (the "2" in the 32nd hexadecimal character).

A record of this type was printed out each time there was a change in the ECD state of a platform and each time there was a change in a platform's workload of required tasks. In figure H-25 one can read from the first line of each record for the ARV at 8141.230 that it was working task number 11 (because of the Boolean 1 in the next to last field of the first line) but not task numbers 4 and 9 (both of their records had Boolean 0 in the corresponding position) and that it had sufficient capability to work tasks 4 and 9 (as indicated by the 1 in the final field of line 1) but not task 11 (the Boolean 0 in the last field of line 1).

The information in this file allowed the health meter postprocessor to determine which task descriptions should be gray, green, or red for each platform. It also allowed the human analyst to explore reasons for task failure and determine which components were contributing to it most often. It was tedious to examine this file by reading it, so it was common to export it to a spreadsheet for easier searching; however, that was beyond the scope of our description in this section.

This file also went through some minor reformatting between the SBM and the health meter postprocessor. The reformatting consisted of the same renumbering of the OWNFOR platform as described for the health history file in the last section.

H.5 Correlation File

The final file written by the SBM was the file of statistics generated from tallies kept during the model run. This was called the correlation file because the majority of the statistics were two-by-two correlation tables. Other statistics printed out in the file are also described here.

During the execution of the simulation, data was collected at regular intervals for statistical calculation and reporting at the end of each run.* During the demonstration, the data collection was performed every 60 s beginning 30 s after initiation of the vignette (at the midpoint of every minute of simulated time).

The first five types of data consist of correlations and related information:

1. Mission success/failure correlated with task pass/fail for every task type.
2. Mission success/failure correlated with ECD (degradation in platform capability) for each ECD for every platform type.

*If the model run included multiple replications of the vignette, statistical calculations represented the entire set of replications. If one wanted statistics for a single replication of the vignette, it was simple to run only one replication.

3. Mission success/failure correlated with component state (functional/dysfunctional) for every component type on every platform type.
4. Task pass/fail correlated with ECD for each ECD for every platform type.
5. Task pass/fail correlated with component state for every component on every platform type.

The correlation format (and related information) printed was similar for all five types of data. First, there was a label telling whether it was the "mission" or a specific "task" vs. a specific task, ECD, or component. This was followed by a line giving the probability (as estimated from the collected data) of the mission or task failing given that the task was failing, the ECD was present, or the component was dysfunctional.

Next was a two-by-two table of the raw sample data that was headed by a line giving the sample size (which should equal the sum of the four table entries). If one considered a hierarchy from high to low consisting of mission, task, ECD, and component, then the two columns correspond to the states in the higher level in the hierarchy, and the two rows to the two states in the lower level.

The sample mean and standard deviation for Y (higher level in hierarchy, mission, or task) and for X (lower level, task, ECD, or component) were then printed out. Then the sample covariance was printed out, followed by the sample correlation (if it was undefined because one or both of the standard deviations was zero, then the correlation was not printed). An example of each of the five correlation-type output cases now follows.

It may help in reading these tables to keep a generic template in mind, as shown in table H-3. X_{00} was the number of times the higher level was in a failure mode (failing mission or task or lacking capability) and the lower level was also in a failure mode (failing task, lacking capability, or component broken); X_{01} was the number of times the higher level was in a success mode (mission or task succeeding or capability functional) and the lower level was in a failure mode; X_{10} was the number of times the higher level was in a failure mode and the lower level was in a success mode (task succeeding or capability intact or component functional); and X_{11} was the number of times the higher and lower levels were in a success mode.

Table H-3. Correlation table template.

—		**Higher Level (Y)**	
		Fail (Lack Capability) 0	**Pass/ Success (Have Capability) 1**
Lower Level (X)	**Fail (0)**	**X_{00}**	**X_{01}**
	Pass (1)	**X_{10}**	**X_{11}**

The estimated (sample) probability of higher level in failure mode given lower level in failure mode was then simply $X_{00} / (X_{00} + X_{01})$.

The sample mean of X was simply $(X_{10}+X_{11})$ divided by the total sample $(X_{00}+X_{01}+X_{10}+X_{11})$, and the sample mean of Y was simply $(X_{01}+X_{11})$ divided by the total sample. They were the means of the marginal variables when treated as binary (0 or 1) random variables. The estimated (sample) standard deviation for X and Y was similarly calculated from the marginal variables.

H.5.1 Mission vs. Task Correlation

For each data collection, the SBM recorded the pass/fail state of all the currently required tasks for all platforms in the OWNFOR and the current assessment of mission success/failure in the absence of reallocation of resources. This allowed the model to calculate and print out correlation information, as shown in table H-4 for each mission vs. task.

Table H-4. Mission vs. task.

Mission versus task 24 LSI A1.5.2 Occupy an Attack/Assault Position ART 2.5.2 by C2V
P (mission failing /task failing) = 0.960744
Sample size = 7580
Raw data

930	38
4966	1646

	Mean	Standard deviation
X =	0.8722960	0.3337600
Y =	0.2221640	0.4157010

Covariance of X and Y = 0.0233581
Correlation of X and Y = 0.1683530

The first line noted the output as a correlation between mission and specific task failure. The task was identified by number and descriptive alphanumeric identifier. The second line gave the fraction of time the mission was failing (as assessed without resource reallocation) when the task was failing. Then four lines reported the raw data collected. The sample size was 7580 (the data were collected every minute for 10 replications and for two C2V type platforms).

Next, a two-by-two table was read as follows. On 930 occasions both the task and the mission were failing; on 38 occasions the task was failing but the mission was not; on 4966 occasions the mission was failing but the task was succeeding (so failures at other tasks were causing mission failure); and on 1646 occasions both the mission and the task were succeeding.

The raw data resulted in a sample mean and standard deviation of 0.872 and 0.334, respectively, of task success (considering task success as a Boolean random variable with 0 representing failure and 1 representing success). Considering the mission success/failure as a similar Boolean random variable yielded a mean of 0.222 and a standard deviation of 0.416.

If possible, the covariance and correlation of X and Y were calculated and printed.* In this case, the correlation was 0.168, indicating that there was not a very strong connection between mission success and success at this particular task.

H.5.2. Mission vs. ECD Correlation

The data collection also supported calculating the correlation between mission success and whether or not each element of capability degradation had occurred. Table H-5 was an example from the correlation file of one such case.

Table H-5. Mission vs. ECD.

Mission vs. ECD: state 2 (m3) on platform type 3 (c2v)		
P (mission failing /ECD failing) = 0.874172		
Sample size = 9600		
Raw data		
132 19		
7562 1887		
	Mean	Standard deviation
X =	0.9842710	0.1244260
Y =	0.1985420	0.3989020
Covariance of X and Y =		0.0011437
Correlation of X and Y =		0.0230434

In this case, the data was for mission failure vs. ECD M3 on platform type C2V (which included both the C2V and its backup). With a sample size of 9600 (over 10 replications again), the instantaneous estimate of mission failure occurred with a relative frequency of 0.874 when the ECD M3 was in effect. From the table of raw data, there were 132 cases where the mission was failing and the ECD was in effect, 19 where the mission was succeeding and the ECD was in effect, 7562 where the mission was failing but the ECD was not in effect, and 1887 where the mission was succeeding and the ECD was not in effect.†

Treating the conditions as Boolean random variables, the sample statistics results were a mean of 0.984 and standard deviation of 0.124 for M3 not occurring and a mean of 0.198 and standard deviation of 0.399 for mission success. The covariance and correlation values of the two random variables were so low that one should not expect this ECD to account for many of the mission failures. This was not surprising given how rarely this ECD happened.

*There were cases in which some of these statistics would be undefined.

†There is a change of convention here. Since an ECD occurring means the loss of a capability, one could either use true to mean that the ECD had occurred or that the capability was still operational (i.e., the ECD has not occurred). The latter convention is used.

H.5.3 Mission vs. Component Correlation

Dropping down from the ECD level to the component level, one could again calculate statistics from the collected data and attempt to draw conclusions. Intuitively, one should not expect strong correlations between mission failure and a single component. Table H-6 shows an example for this kind of data and the corresponding statistics.

Here the two random variables were mission success or failure and the component states functional/dysfunctional of component type 150 on an ARV-type platform. There were 14400 samples, and in all of the 96 cases where component 150 was dysfunctional, the mission was estimated to be failing. Table H-6 shows that there were 96 samples where the mission was failing and the component was dysfunctional, no samples where the mission was succeeding and the component was dysfunctional, 11445 samples where the mission was failing but the component was functional, and 2859 samples where the mission was succeeding and the component was functional.

Table H-6. Mission vs. component.

Mission vs. component 150 on arv		
P (mission failing /this component dead) = 1		
Sample size = 14400		
Raw data		
96	0	
11445	2859	
	Mean	Standard deviation
X =	0.9933330	0.0813770
Y =	0.1985420	0.3989020
Covariance of X and Y =		0.0013236
Correlation of X and Y =		0.0407748

The mean and standard deviation for the Boolean random variable indicating that the component was functional are 0.993 and 0.081, respectively. For the random variable indicating that the mission was succeeding, they were respectively 0.199 and 0.399.

The low correlation value of 0.041 indicated that there was not much explanatory value for mission success or failure in this component's state considered alone. It appeared to be necessary for mission success but definitely not sufficient.

H.5.4 Task vs. ECD Correlation

The data collected were also used to calculate statistics for task success vs. each ECD. An example is shown in table H-7.

Table H-7. Task vs. ECD.

Task vs. ECD: Task no. 0 ART 3.3.1.1 Conduct Surface to Surface Attack for platform type 5 (nlos_c)
Task 0 vs. ECD: state 0 (m1) on platform nlos_c
P (task failing /this degradation) = 0

Sample size = 1200
Raw data

0	71
130	999

	Mean	Standard deviation
X =	0.9408330	0.2359360
Y =	0.8916670	0.3108010

Covariance of X and Y =	–0.00640972
Correlation of X and Y =	–0.0874102

Here the task was to conduct a surface-to-surface attack with an NLOS-C platform; the ECD was M1 (reduced forward speed). This degradation never caused a task failure in 1200 samples. The collected data showed no cases where both the task failed and the ECD was in effect (the capability lost), 71 cases where the task succeeded and the ECD was in effect (capability lost), 130 cases where the task failed and the ECD was not in effect (capability intact), and 999 samples where the task was successful and the ECD had not taken effect (capability intact).

Treating both task success/failure and ECD in effect/not in effect as Boolean random variables yielded a mean and standard deviation of 0.941 and 0.236, respectively, for absence of ECD (retention of capability) and a mean and standard deviation of 0.892 and 0.311, respectively, for task success (really of having the instantaneous capability to work the task).

In this case the correlation was negative. However, it was so small in absolute value that one should not draw any conclusion; having reduced forward speed (the M1 ECD) would not necessarily help improve task success. Indeed one would be interested in examining the 130 samples where the task failed even when the capability was intact (ECD not in effect). There might have been cases where the NLOS-C was required to be capable of firing without any requirement to move.

H.5.5 Task vs. Component Correlation

The example shown in table H-8 used data collected for correlating task success/failure with component function/dysfunction.

Table H-8. Task vs. component.

Task vs. component 82 on platform uav		
P (task failing /this component dead) = 1		
Sample size = 200		
Raw data		
20 0		
15 165		
	Mean	Standard deviation
X =	0.0900000	0.3000000
Y =	0.8250000	0.3799679
Covariance of X and Y =		0.0825000
Correlation of X and Y =		0.7237470

In all cases where component 82 on a UAV-type platform was dysfunctional, the task (task 6 was "Report Enemy Information") was failing. This was based on 200 data samples collected during the simulation run.

Per table H-8, in 20 samples both the task failed and the component was dysfunctional, in 0 cases the task was successful and the component was dysfunctional, in 15 cases the task was failing with the component functional, and in 165 cases the task was successful and the component was functional.

The mean and standard deviation for the Boolean random variable of component being functional were respectively 0.9 and 0.3. For the Boolean random variable of having sufficient capability for task success, they were 0.825 and 0.380.

This time the correlation was fairly high at 0.724. This is not, of course, high enough for functioning of component 82 to be a necessary and sufficient condition for task success, but it clearly indicated that component 82 was an important component on the UAV for the capabilities needed for these tasks. The explanatory value for UAV mission success would be worth further exploration.

H.5.6 Relative Frequency of Occurrence of ECD by Platform Type

After reporting all of the correlation tables of the five types just described, the correlation file continued with some other statistics. The first was a table that showed for each platform type the fraction of its time in the vignette that a platform of that type was found to have a given ECD in effect (that was, a given loss of capability). An example is shown in table H-9.*

*The table has been truncated to avoid lines wrapping. In the full version, there would be 31 columns corresponding to the ECDs.

Table H-9. Time spent in a degraded capability state.

	M1	*M2*	*M3*	*M4*	*F1*	*F2*	*F3*	*F4*	*F5*	*A1*	*A2*
ARV	0.04	0.08	0.01	0.06	—	—	—	—	0.02	0.00	0.00
	0.12	0.15	0.04	0.10	—	—	—	—	0.05	0.04	0.03
UAV	0.11	0.23	0.11	0.25	—	—	—	—	—	—	—
	0.25	0.27	0.25	0.25	—	—	—	—	—	—	—
C2V	0.04	0.02	0.02	0.12	0.00	—	—	—	—	—	—
	0.13	0.12	0.02	0.12	0.00	—	—	—	—	—	—
NLOS-C	0.06	0.10	0.03	0.08	—	0.01	0.10	0.09	0.04	—	—
	0.12	0.16	0.04	0.08	—	0.06	0.12	0.12	0.06	—	—

Each table column corresponded to an ECD, and each pair of rows corresponded to an OWNFOR platform type. The top number in each column-row intersection was the fraction of the vignette time that a platform of the given type spent in each ECD. For example, looking at the column headed M2 and the row headed C2V, one can see that the two platforms (the C2V and its backup) of type C2V spent an average of 2% of their time in state M2 (reduced maneuverability) over the 10 replications of the vignette in this run. The 0.12 immediately below the 0.02 in the M2 column for the C2V indicated that when implied states are included, the percentage of time spend with an M2 ECD in effect was 12%.*

This table allowed the analyst to see which ECDs were likely to cause frequent force degradation. Again, as with any of these statistical measures, it was usually only an indicator of possible influence and not a guarantee of a causal relationship.

H.5.7 Task Pass/Fail Rate

Table H-10 shows an excerpt from the task pass/fail rate collected over the replications of the vignette in a run (10 replications in this case). There were 31 different tasks in the vignette. They were numbered 0–30 by the program (first number in each row), but each had its own authoritative number assignment and descriptor: LSI, Army Universal Task List, or universal joint task list. Each row also included a count of the number of times the task was sampled and the platform had sufficient capability to perform the task (a "pass") and the number of times the task was sampled and the platform did not have sufficient capability to perform it (a "fail"). These counts were also converted to fractions.

*Recall that an ECD could occur either because some dysfunctional component caused a fault tree to be cut or because the ECD was implied by the occurrence of another ECD. In the example of the M2, it was implied whenever an M4 occurs.

Table H-10. Pass/fail rates by task.

Task No.	Pass	Fraction	Fail	Fraction	Samples for This Task by Indicated Platform:
0	1070	0.8917	130	0.1083	ART 3.3.1.1, conduct surface-to-surface attack by NLOS_C
1	995	0.9950	5	0.0050	ART 7.2.5, disseminate common operational picture and execution information by C2V
2	1560	0.7800	440	0.2200	LSI A1.2, conduct tactical maneuver ART 2.2 by C2V
3	7323	0.9687	237	0.0313	LSI A1.2 conduct tactical maneuver ART 2.2 by NLOS_C
4	95	0.9500	5	0.0500	LSI A1.6.2.1.1.4.3, report enemy information by ARV
5	900	1.0000	0	0.0000	LSI A1.6.2.1.1.4.3, report enemy information by ARV
6	165	0.8250	35	0.1750	LSI A1.6.2.1.1.4.3, report enemy information by UAV
7	198	0.7920	52	0.2080	LSI A1.6.2.1.2.2.1, launch and recover UAV by UAV
8	5190	0.6793	2450	0.3207	LSI A1.6.2.1.2.2, fly UAV mission by UAV
9	3555	0.9186	315	0.0814	LSI A4.2.1, conduct tactical reconnaissance by ARV
10	959	0.7797	271	0.2203	LSI A4.2.1, conduct tactical reconnaissance by UAV
11	9399	0.8943	1111	0.1057	LSI A4.2.2, conduct surveillance by ARV

Note that these data were reported by platform type, not by individual platform. Also, the samples were taken only for times when the task was required of a platform. For example, task number 8 as an example was sampled 7640 times (5190 + 2450) during the vignette. Since this particular run consisted of 10 vignette replications, each replication contributed 764 samples. Since this task was performed by the UAV type of platform of which there were three in the vignette, each UAV was sampled an average of 254.67 times during the run of the vignette to determine whether it had the capability to perform this task.* This task failed almost one-third of the time (0.3207 to be exact) in the vignette because on many replications a catastrophic failure or damage would take out one of the UAVs.†

It was also important to understand that these were time-weighted statistics in that a task may be required of a platform for 60 min at one time in the vignette and for only 5 min at a later time. The former requirement would be sampled 60 times whereas the latter only 5 times; consequently, having capability throughout the former would appear more important than the latter as far as this statistical summary was concerned.

It was easy to see from the output that some tasks were sampled rather infrequently, whereas others were almost constantly required of their platform type. Thus the table not only gave a report on the time-weighted pass/fail capabilities of the platforms, but also on how long they

*The samples were taken every minute to determine whether the sample size was reasonable. That number of samples would correspond to just under 4 h and 30 min per UAV (4 h, 14 min, 40 s), which was a reasonable time for each of them to be flying a UAV mission (two of them were already in the air when the vignette began, so the actual mission duration for them was slightly longer than this indicated).

†Recall that for a UAV, a total immobilization means that it crashed, not that it stopped and awaiting repair like a ground platform.

were required during the vignette. Of course, the duration of time a task was required does not necessarily indicate its importance to mission success. A routine task may be required for the entire duration of the vignette, whereas an important message may only require communications capability for a minute fraction of the vignette; however, they may both be essential to the ultimate success of the mission.

If the model had used real data instead of fictitious data, this table would allow the analyst to see which tasks were failed for the most time. That information might lead to modification of the tasks composing the mission to reduce the risk of mission failure or to platform design modifications to improve capability survivability.

H.5.8 Mission Failure Causes

The previous output showed time-weighted task pass/failure rates but did not tie that information to mission success or failure. The next section of output on the correlation file showed the actual time duration that each critical task was failing and seemed to be causing mission failure.*

The same task numbers were used as in the previous output, but this time the total number of failures (samples where the platform was found to have inadequate capability to perform the task) was recorded. For example, in table H-11, task 7 failed on 2185 samples over the 10 replications of the vignette. This time, however, a failure means that the task was not only required of the platform but that it was mission critical (during the phase of the vignette when the sample was taken). If one or more platforms of the relevant type were incapable of performing the task at a time when it was considered critical, it counted as a failure for this tally; if all platforms of the relevant type were capable of performing the task or if the task was not considered critical at the sampled time, then it was not counted as a mission failure. During 0.4552 of the vignette duration, there were one or more UAVs incapable of performing task 8 when it was critical. Note that this fraction is taken over the entire vignette duration, and that at many times during the vignette there are multiple critical tasks being failed.†

*Whether it actually caused mission failure in the end was another matter since the task might have been scheduled for a 30-min interval but was completed in the first 5 min, followed by a failure throughout the final 25 min of the same interval. This would have been scored as significant failure time even though the task was successful and the mission unaffected by that failure.

†In this example, adding up the time fractions will exceed 1.00.

Table H-11. Mission failure causes without resource adjustment.

Task 0	Causing mission failure on	40	Samples	0.00083	of vignette duration
Task 1	Causing mission failure on	5	Samples	0.0020	of vignette duration
Task 2	Causing mission failure on	405	Samples	0.0844	of vignette duration
Task 3	Causing mission failure on	51	Samples	0.0106	of vignette duration
Task 4	Causing mission failure on	5	Samples	0.0010	of vignette duration
Task 5	Causing mission failure on	35	Samples	0.0073	of vignette duration
Task 6	Causing mission failure on	52	Samples	0.0108	of vignette duration
Task 7	Causing mission failure on	2185	Samples	0.4552	of vignette duration
Task 8	Causing mission failure on	286	Samples	0.0596	of vignette duration
Task 9	Causing mission failure on	271	Samples	0.0565	of vignette duration
Task 10	Causing mission failure on	1107	Samples	0.2306	of vignette duration
Task 11	Causing mission failure on	1774	Samples	0.3696	of vignette duration
Task 12	Causing mission failure on	15	Samples	0.0031	of vignette duration
Task 13	Causing mission failure on	57	Samples	0.0119	of vignette duration

The tally reported in the mission failure rate table (table H-11) did not take into account redundancy of capability across platforms—one of the benefits of the system of systems. When one platform was failing a mission-critical task but there was another capable platform available to perform the task, resources could be reallocated. With resource adjustment the frequency of failing mission-critical tasks was much lower. Table H-12 shows the results when redundancy and reallocation were taken into account.

Table H-12. Mission failure causes with resource adjustment.

Task 0	Causing mission failure on	12	Samples	0.0025	of vignette duration
Task 2	Causing mission failure on	35	Samples	0.0073	of vignette duration
Task 3	Causing mission failure on	2	Samples	0.0004	of vignette duration
Task 19	Causing mission failure on	35	Samples	0.0073	of vignette duration
Task 20	Causing mission failure on	35	Samples	0.0073	of vignette duration
Task 22	Causing mission failure on	35	Samples	0.0073	of vignette duration
Task 24	Causing mission failure on	66	Samples	0.0138	of vignette duration
Task 25	Causing mission failure on	259	Samples	0.0540	of vignette duration
Task 28	Causing mission failure on	10	Samples	0.0021	of vignette duration

Again the count was over 10 replications of the vignette, but it was incremented only when the sampling found a time when the task was critical, the platform had inadequate capability to perform the task, and there were not enough available capable alternative platforms to take over the task. Both the frequency and fraction of the total vignette duration were much lower than the previous mission failure counts when tasks were failed with resource adjustment. Indeed, now task 7 did not show up as being failed. This was due to sufficient alternative resources to perform the task.

In the SBM the dynamic resource reallocation was extremely crude. There was a required number of platforms of each type below which the number capable of performing each critical task must not fail. As long as the count did not fall below that threshold, the resources were considered sufficient that an adjustment could be performed. A fully dynamic resource reallocation taking into account time to switch tasks and other factors was planned as a project for a follow-on model.

Statistics on mission failure are shown in table H-13. The first line reported that on 3847 samples (minutes) over the ten replications, one or more critical tasks were failing by one or more platform and that this was happening 0.8015 of the vignette duration. When adjustment (reallocation) of resources was taken into account, the corresponding numbers drop to 376 and 0.0783, respectively. The great improvement when resource reallocation was allowed shows the clear benefit of avoiding single points of failure in the mission planning.

Table H-13. Mission failure.

Without adjustment, mission failing on	3847 checks	0.80 of time
With adjustment, mission failing on	376 checks	0.08 of time
Commander's intent achieved on 8 of 10 replications		

Finally, the specified commander's intent was determined. Recall the commander's intent was achieved if at least one of the C2V or its backup, at least seven of the MCSs, and at least four of the NLOS-Cs were functional at the end of the vignette and had reached their objective in the case of the MCS and C2V (or backup). This was achieved for this run on 80% (8 out of 10) of the replications. This number did not track too well with the mission failure percentages on the previous two lines because a failure or combat damage late in the vignette might contribute little to the "failure duration" measure yet still contribute to failing to meet the commander's intent. Still, taken over a number of different runs with different failure, combat damage, and repair rates, one would see that the commander's intent was generally met more often in cases with low component failure rates.

INTENTIONALLY LEFT BLANK.

Appendix I. A Rigorous Way to Reason About Platform-Level Readiness

The Missions and Means Framework (MMF) provides a structured way to express and explore the interrelationship between the requirements imposed by a mission and the capabilities offered by a materiel system. In so doing, the MMF also makes it possible to assess both the extent to which the system is capable in a particular state and the sufficiency of that capability state to the demands of the current mission context. We lay out the details of an approach to that assessment and illustrate it with an application—a notional instrument-panel display to convey a running impression of a platform's readiness. One benefit of the approach is that it suggests a new, higher-level conception of vehicle prognostics and diagnostics that is also much more relevant to the operational Warfighter than the traditional conception, which is focused on maintenance and reliability.

I.1 Mathematical Foundations

This assessment approach makes use of the mathematical objects called *partially ordered sets*. We begin our presentation of the approach with a review of their basic properties. Colloquially, a partially ordered set is a collection of objects together with a relation (the order) under which some pairs of the objects are related, but not necessarily every pair (hence it is partial). Formally, a partially ordered set, often shortened to *poset*, is an ordered pair $(X, \blacktriangleleft)$ consisting of a set X together with a relation $\blacktriangleleft$ on X that is reflexive, antisymmetric, and transitive.

The concept of partial order may be understood as a generalization of the standard *less-than-or-equal-to* order on the real numbers, which gives the poset $(\mathbf{R}, \leq)$. In fact, typical practice when dealing with any general, abstract poset is to denote its relation not by $\blacktriangleleft$, but by $\leq$. The partial-order concept also generalizes the containment, or *subset-of*, relation $\subseteq$ on a collection of sets. Of course, under *less-than-or-equal-to* every pair of real numbers is related, or *comparable*. But concerning containment among sets, there are many pairs that are *incomparable*—for instance, neither of $\{b, d\}$ and $\{c, d, t\}$ is a subset of the other. Those partial orders that, like $\leq$ on the reals, have no incomparabilities are called *total orders*.

The standard way to represent any poset $(X, \leq)$ graphically is the Hasse diagram. This is a drawing in which the elements of X are shown as dots and the comparabilites in $\leq$ are shown by upward paths of line segments linking the dots.* As an example, consider the *divides* relation, $|$, on the first 10 positive integers: the Hasse diagram of the poset $(\{1, \ldots, 10\}, |)$ is shown in figure I-1.

*A technical detail—for a Hasse diagram, the only pairs of dots (distinct elements x and y of X) between which we draw segments are those for which $x \leq y$ and there is no third element z of X with $x \leq z \leq y$. These segments constitute what is called the *transitive reduction* of $\leq$. In the particular case of a poset, the transitive reduction is also called its *cover* relation.

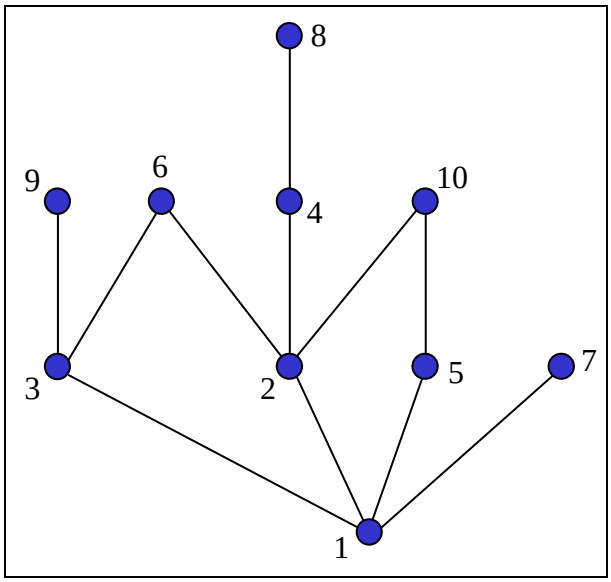


Figure I-1. The Hasse diagram of the integers {1, …, 10} ordered by *divides*.

I.2 Representing the State of a Platform's Capability

To achieve sufficient precision and rigor, we express the state of a platform's capability at any instant by enumerating its shortcomings. Specifically, for any analysis we define a suitable collection Δ of relevant flaws called *elements of capability degradation* (ECDs) and express the platform's capability state as that subset of Δ that includes precisely those flaws that the platform suffers. In practice, it is typically appropriate to partition Δ into categories along the lines

$$\Delta = \{\text{mobility ECDs}\} \cup \{\text{firepower ECDs}\} \cup \{\text{communication ECDs}\} \cup \ldots$$

I.3 Comparing Capability States

We wish our technique for assessing the mission sufficiency of a platform's capability state to be a phased approach, addressing first that portion of readiness that is independent of mission context and then incorporating the particulars of individual missions. But for arbitrary sets $A, B \subseteq \Delta$ of ECDs there is, in general, no way to make absolute assertions of the form "A is a state of lesser fitness than B." Just as biological fitness is defined only with respect to some ecological circumstance, so, the potential states of a platform can be compared for relative ability only in the context of an operational mission. On the other hand, there is a simplifying assumption that allows us to draw many mission-independent inferences in a logically justifiable manner: because inflicting flaws on a platform never enhances its capabilities, it is reasonable by way of approximation to consider A to be a generally less fit state than B whenever $A \supset B$. In other words, this means that if state A comprises all the ECDs of state B and then some, then one is safe in considering A to be worse than B.

There are two points about this simplifying assumption that should not be overlooked. First, like any other simplification, it is imperfect: for any states A and B with $A \supset B$ one can typically imagine operational contexts in which acquiring the additional ECDs that take it to A from B does not render a platform any less fit than it already was when in B. But it is not clear how significant a problem this imperfection would be in practice. Second, the only reason that the assumption justifies concluding that A is worse than B is that A is in fact a proper superset of B. In particular, concluding that A is worse than B is *not* justified by A's merely containing a larger number of ECDs than B does. To see that simply counting ECDs is insufficient for comparisons of fitness, consider that—again depending on the operational context—one might have far more trouble from a single unfortunately chosen flaw than from an entire collection of flaws that are less relevant to the immediate needs.

This comparison of all of a platform's conceivable capability states based on containment is, like all containment relations, a partial order.* Two such states are incomparable under the *no-better-than* relation $\supseteq$ if and only if each includes at least one ECD that the other does not. Because it is a partial order, we can apply mathematical tools from order theory, which buys us logical rigor.

I.4 Attainability of States

As we've seen, the simplifying assumption allows us to address those questions about platform capability that are independent of mission context. Besides dividing up our problem neatly, does the assumption offer any advantage that compensates for the imperfection noted above? There is indeed value in exploring the interrelations among the possible capability states themselves.

Recall that each state is a set $S \subseteq \Delta$ of ECDs, so the poset we are interested in is $(\Sigma, \supseteq)$, where Σ is the set of all states that the system can possibly attain. Which are the attainable states? If Δ comprises n ECDs, then aren't all 2^n subsets of Δ possible? No, there are two types of constraints that, in practice, substantially restrict which states the system can attain. To illustrate these constraints, consider a notional soldier ensemble system. To address the system's posture capabilities, it might be appropriate to include in Δ such ECDs as

p_1 = diminished left-leg strength,

p_2 = diminished right-leg strength,

p_3 = no left-leg strength,

p_4 = no right-leg strength,

*At this point we police up a small bit of definitional laxness. Technically, the $\supset$ relation we have been discussing as the *worse-than* relation on capability states is not a partial order because it is not reflexive—no set is a proper superset of itself, nor do we want to consider any set to be worse than itself. Rather, $\supset$ is a *strict partial order*, which is to say a relation that is irreflexive and transitive. For the purists, its reflexive closure, $\supseteq$, which we might call the *no-better-than* relation, *is* a partial order. This minor distinction is not particularly important for our present purposes.

p_5 = diminished balance, and

p_6 = no upright stance.

Constraints of the first type are semantic in nature. As an example, note that there is an implication intrinsic to what it means to lose *some* strength in a leg vs. losing *all* strength in that leg. The implication might be expressed as "it is nonsensical to contemplate a soldier's being in a state that contains p_3 but does not also contain p_1 (or contains p_4 but not p_2)." The type-two constraints result from the details of each system's design. As an example, suppose that a bipedal stance demands at least some strength in each leg and some balance, while a one-legged stance demands full strength in either leg and full balance. If an upright stance, in turn, requires standing bipedally or on one leg, then no state that includes p_5 and either of p_3 and p_4 is attainable unless it also includes p_6. But that constraint is an artifact of the design of the system: had the design provided a tripedal capability, say by incorporating into the ensemble a walking stick or crutch, then both states $\{p_3, p_5\}$ and $\{p_4, p_5\}$ would be attainable.

I.5 Mapping Capability States to the Real Line

Even though the set Σ of attainable states may be far smaller than the collection 2^A of all conceivable states, the full structure of our poset $(\Sigma, \supseteq)$ may still be too cumbersome to contend with for some applications. It is hard to imagine, for instance, that a Hasse diagram would be a particularly effective format to convey the state of an aircraft to its pilot in combat. But depending on the requirements of the application, there are techniques for simplifying the data to produce useful and usable displays.

As an example we suggest a simple technique to convey, by analogy with a conventional dashboard fuel gauge, a rough sense of how much of a system's original, built-in capabilities remain available as a mission proceeds. Of course, an analog fuel gauge allows comparisons of states that are (real-valued) quantities ordered by the total order $\leq$, so they correspond perfectly to positions on a number line. By contrast, capability level is, in general, not a scalar concept that admits a total order. We overcome this challenge by augmenting the *no-better-than* relation, $\supseteq$, of our poset with enough additional comparabilities to produce a partial order that, although not necessarily total, corresponds to (possibly duplicated) positions on a number line. We thus obtain what is called a *weak extension* of $\supseteq$.*

*Such a *weak* poset $(X, \leq_w)$ is significantly more regular in structure than is typical of posets in general. The weak order partitions X (in our case, $X = \Sigma$) into a number of parts $X_1, \ldots, X_p$ such that all the members (our states) of any part are incomparable to one another but comparable to every member of all the other parts. Thus if X is finite (ours is), the partition gives rise trivially to an integer labeling λ: $X \to \{1, \ldots, p\}$ that assigns to each element of X the sequential number of its part in the partition such that for any elements x and y of X we have $\lambda(x) \leq \lambda(y)$ if and only if $x \leq_w y$. An element's label may be interpreted as specifying both the part it belongs to and its position on the real line.

Each of the positions on the number line may be thought of as a bin containing a maximal collection of attainable states that are of roughly similar capability levels. More rigorously, since any distinct states S_1 and S_2 in a bin are incomparable under $\supseteq$, each includes at least one ECD not in the other, so absent any mission details and the resulting sensitivities to the particular ECDs in question, one cannot conclude that either of S_1 and S_2 is a lesser fitness than the other.

The positions along such a capability gauge range from the worst case at the "E" (for *empty*) end of the scale—which corresponds to the very state $S_E = \Delta$ where every relevant capability has suffered complete degradation (when $\Delta \in \Sigma$, so that state is attainable)—all the way to the system's full original capability, corresponding to the state $S_F = \{\}$ that contains no ECDs.

I.6 Factoring in the Mission

The particular tasks that must be performed or performable by a system at any point during the execution of a mission impose the requirement for a set of capabilities that are demanded of the system at that point in the mission. That set of capabilities that are required may also be seen complementarily as a set of ECDs that are forbidden. Either way, for any state S in Σ, the tasks required at time t in the mission lead directly to an unambiguous binary scoring $\alpha(S, t)$ of the instantaneous adequacy of S expressing whether the system, were it in state S, could accomplish the tasks current at time t in the mission.

We incorporate all of these ideas into a single display design. For each of the ECD categories (e.g., mobility, communication, …) we generate a capability-gauge bar broken into segments that correspond to the state bins of roughly homogeneous capability levels. As the mission proceeds, we move each gauge's needle from segment to segment so that it always indicates the bin containing the system's current state. This provides a dynamic indication of how the system's instantaneous capabilities fit within the complete range of capability levels that any system of such design and construction could possibly attain. Finally, behind the needle, we color each segment of the gauge according to the instantaneous adequacy scoring α of the corresponding states. If every state in the bin would be adequate to accomplish the current task(s), we color the segment green. If no state in the bin suffices for the current tasks, we color the segment red. And if the bin contains some states that are adequate and others that are not, we color the segment yellow. By itself, this coloring of a gauge's segments provides a dynamic indication of how stressing the demands of the current tasks are on the overall design of the system. In combination with a gauge's needle, the segment coloring provides a dynamic indication of whether the system is currently capable of performing the tasks required of it and by how much it either exceeds or falls short of its capability requirements.

This display design is certainly feasible for use in simulations and analyses to reason about the dynamic mission-relevant health of a system. Furthermore, the level of computerization in today's operational environment should make such a display practical to implement in the real system as well. If the system's design includes sufficient instrumentation to sense the operating status of its mission-critical components over time, then system capabilities (and ECDs) should

be available in real time. And because today's processes for mission planning are, like contemporary training designs, thoroughly based on tasks, an automated system, whether on board or behind the lines, could have available to it the tasks that the commander currently expects of it.

I.7 High-Level Prognostics and Diagnostics

The same logic that allows us to provide a display supporting comparisons of current capabilities to current requirements could also be used to support reasoning about the remainder of the mission. Beyond the tasks that a system is engaged in currently, the planning process will also have worked out all the follow-on tasks up to the completion of the mission. So a mechanism could extrapolate, however might be appropriate, from the system's current state to its capabilities at any later time and compare those capabilities to the tasks that will be expected of the system at that future point in the mission. This would make possible a form of prognostication dealing not in the terms natural for a maintenance or logistics perspective, but in the language of the operator. Instead of reporting, say, that some track pad has another 1500 km of service life remaining, it could report that the system's reduced capability in digital communication will allow it to proceed to its specified firing position by the time required for synchronization, but will prevent it from properly processing calls for fire and thus effectively delivering fire.

In the logistics and maintenance contexts to which prognostics has conventionally been applied, this MMF-based way of handling readiness offers a rigorous and automated approach to the prioritization, scheduling, and resourcing activities while providing a means for logistics staff officers to support the force commander in assessing the mission impact of current and projected status. Leveraging planners' subjective, experience-based processes with this more rigorous approach that explicitly addresses the linkage between materiel state and mission effectiveness, allows optimizing investment of effort to maximize the benefit to the supported commander.

List of Symbols, Abbreviations, and Acronyms

ALT	alternative (short for alternative or backup C2V)
AP	attack position
APC	armored personnel carrier
APS	Active Protection System
ARL	U.S. Army Research Laboratory
ARV	armored robotic vehicle
ATEC	U.S. Army Test and Evaluation Command
AUTL	Army Universal Task List
AWK	Aho Weinberger Kernighan (scripting language named after its authors)
BCT	Brigade Combat Team
BDA	battle damage assessment
C2V	command and control vehicle
CAB	Combined Arms Battalion
CCIR	Commander's Critical Intelligence Requirements
CEP	circular error probability
CJTF	Combined Joint Task Force
COA	course of action
COP	common operating picture
CSVG	component state vector generator
DOD	Department of Defense
DOT&E	Director, Operational Test and Evaluation
DOTMLPF	doctrine, organization, training, materiel, leader development, personnel, and facilities
DRC	Dynamics Research Corporation
EA DUNK	engagement area DUNK

ECD	element of capability degradation
ESS	effectiveness, suitability, and survivability
FCS	Future Combat System
FOV	field of view
FS	functional skeleton
FTTS	future tactical truck system
JCIDS	Joint Capabilities Integration and Development System
JOA	Joint Operations Area
LER	loss-exchange ratio
LSI	lead system integrator (for the FCS program)
M/F LoF	mobility or firepower loss of function
MAPEX	Army map exercise
MBT&E	mission-based test and evaluation
MCS	Mounted Combat System
MDMP	military decision-making process
MMF	Missions and Means Framework
MOE	measure of effectiveness
MOP	measure of performance
MPI	mean point of impact
MTBF	mean time between failures
MTTR	mean time to repair
MUVES	Modular Unix-Based Vulnerability Estimation Suite (Survivability/Lethality Analysis Directorate's vulnerability/lethality model)
NAI	named area of interest
NLOS-C	Non-Line-of-Sight Cannon
$O_{1,2}$	mapping operator from level 1 to level 2
$O_{2,3}$	mapping operator from level 2 to level 3

$O_{3,4}$	mapping operator from level 3 to level 4
OMS-MP	operational mode summary – mission profile
O&O	Organization and Operation
OPFOR	opposing force
OPORD	operational order
OPLAN	operational plan
ORD	Operational Requirements Document
OWNFOR	own force
S4	System of Systems Survivability Simulation
SA	situation awareness
S&RO	Stability and Reconstruction Operations
SBM	Storyboard Model
SCAP	systems capability analytic process
SITL	Single Integrated Task List
SME	subject matter expert
SoS	system of systems
SPH	self-propelled howitzer
TAI	targeted area of interest
T/C/S	task/condition/standards
TOEL	time-ordered event list
TOC	tactical operations center
TRADOC	Training and Doctrine Command
TTP	tactics, techniques, and procedures
UA	unit of action
UAMBL	Unit of Action Maneuver Battle Lab
UAV	unmanned aerial vehicle
UE	unit of employment

UJTL	Universal Joint Task List
V/L	vulnerability/lethality

NO. OF COPIES	ORGANIZATION
1 (PDF only)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1	DIRECTOR US ARMY RESEARCH LAB IMAL HRA 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL 2800 POWDER MILL RD ADELPHI MD 20783-1197

NO. OF
COPIES ORGANIZATION

1 USARL
RDRL SLE
R FLORES
WSMR NM 88002-5513

ABERDEEN PROVING GROUND

1 DIR US ARMY EVALUATION CTR HQ
TEAE SV
P A THOMPSON
2202 ABERDEEN BLVD 2ND FL
APG MD 21005-5001

3 DIR USARL
(2 HC RDRL SL
1 PDF) J BEILFUSS
P TANENBAUM
RDRL SLB A
M PERRY (PDF only)

NO. OF COPIES ORGANIZATION

1 (PDF only) OFFICE OF THE DEPUTY CHIEF OF STAFF G-4
DALO ZX
J CORCORAN
500 ARMY PENTAGON (1E394)
WASHINGTON DC 20310-0500

1 (PDF only) OFFICE OF THE DEPUTY CHIEF OF STAFF G-3/5/7
DAMO CIA
COL R HOLDREN
4000 ARMY PENTAGON (2E382)
WASHINGTON DC 20310-0400

2 (PDF only) G-8 ARMY STUDY PRGM MGMT
PENTAGON (3E393)
M MARIMAN
D TISON
WASHINGTON DC 20310

8 (PDF only) OFFICE OF THE DIRECTOR OPRTNL TEST AND EVAL (DOT&E)
D DUMA
R SAYRE
S DALY
J STREILEIN
S KOCH
C WARNER
B HALL
T FISHER
1700 DEFENSE PENTAGON (1D548)
WASHINGTON DC 20301-1700

1 (PDF only) OFFICE OF THE DIRECTOR
JOINT ANALYTICAL SUPPORT DIV
OFC OF THE SECRETARY OF DEFENSE COST ASSESSMENT AND PRGM EVAL
J BEXFIELD
1800 DEFENSE PENTAGON (3C117)
WASHINGTON DC 20301-1800

1 ARMY CAPABILITIES INTEGRATION CTR (ARCIC)
DIR ANLYS AND INTEGRATION
950 JEFFERSON AVE
FORT EUSTIS VA 23604-5761

2 CENTER FOR ARMY ANALYSIS (CAA)
DIR CAA AND TECH DIR CAA
6001 GOETHAIS RD
FORT BELVOIR VA 22060-5230

NO. OF COPIES ORGANIZATION

1 DIRECTOR REQUIREMENTS INTEGRATION DIRCTRT ARMY CAPABILITIES INTEGRATION CTR
US ARMY TRAINING AND DOCTRINE CMND (TRADOC)
BG REAGAN
950 JEFFERSON AVE
FORT EUSTIS VA 23604-5770

1 DIRECTOR ANALYSIS AND INTEGRATION CTR US ARMY CAPABILITIES INTEGRATION CTR
ATFC R
A RESNICK
950 JEFFERSON AVE (B950)
FORT EUSTIS VA 23604-5770

1 HEADQUARTERS AMC (HQ AMC)
CHIEF TECHLGY OFFICER
G BOCHENEK
4400 MARTIN RD
REDSTONE ARSENAL AL 35898-5000

1 HQ AMC DIR STRATEGY AND CONCEPTS
M GREY
400 MARTIN RD
REDSTONE ARSENAL AL 35898-5000

1 US ARMY TRAINING AND DOCTRINE CMND (TRADOC)
ATRC W
DR PIPPIN
B1400 MARTIN LUTHER KING
WSMR NM 88002-5502

2 US ARMY TRADOC ANALYSIS CTR
W KRONDAK
P WORKS
255 SEDGWICK AVE
FORT LEAVENWORTH KS 66027

1 TRADOC ANALYSIS CTR
ATRC
P BLECHINGER
255 SEDGWICK AVE
FORT LEAVENWORTH KS 66027-2345

1 JOINT TEST AND EVAL CTR STE 105
DEFENSE INFORMATION SYS AGCY
M LORENZO
7025 HARBOUR VIEW BLVD
SUFFOLK VA 23435

NO. OF COPIES	ORGANIZATION
2	US ARMY TRAINING AND DOCTRINE CMND (TRADOC) ANLYS CTR ATRC WMD C MULLIS K YOUNG B1401 MARTIN LUTHER KING WSMR NM 88002
1 (CD only)	US ARMY TRAINING AND DOCTRINE CMND (TRADOC) ANLYS CTR ATRC PR/S MATUS 255 SEDGWICK AVE FORT LEAVENWORTH KS 66027-2345
3	USARL RDRL SLE G J THOMPSON P DJANG J SMITH WSMR NM 88002-5513
1	USA TACOM AMSTA CSB V M KERR 66501 E 11 MILE RD WARREN MI 48397-5000
3	ORSA CORPORATION W YEAKEL J SHEEHAN R SANDMEYER 1003 OLD PHILADELPHIA RD ABERDEEN MD 21001
1	DYNAMICS RSRCH CORP B BRAY 4716 WENATCHIE TRAIL LIMA OH 45805
3	DYNAMICS RSRCH CORP J LOCHOW M MINCHEW C THURMAN 108 S 5TH ST FORT LEAVENWORTH KA 66048
3	RAND CORPORATION T BONDS J BOON C PERNIN 1200 S HAYES ST ARLINGTON VA 22202-5050

NO. OF COPIES	ORGANIZATION
1	DIRECTOR OF INTEGRATION LOCKHEED MARTIN CORP TEST AND EVALUATION T WISSINK 700 N FREDERICK AVE 182/2A22 GAITHERSBURG MD 20879
1	APPLIED RSRCH ASSO ENGRG SCI DIV J HANES 421 OAK AVE PANAMA CITY FLA 32401
1	APPLIED RSRCH ASSO ENGRG SCI DIV F MAESTAS STE A-220 4300 SAN MATEO BLVD NE ALBUQUERQUE NM 87110
1	RAYTHEON COMPANY B WILSON 528 BOSTON POST RD SUDBURY MA 01776
1	SURVICE ENGRG CO J WALBERT 3700 FETTLER PARK DR STE 401 DUMFRIES VA 22025
1	SURVICE ENGRG CO E EDWARDS 4695 MILLENNIUM DR BELCAMP MD 21017
1	COL(R) J APPLEGET 1411 CUNNINGHAM RD GL 239 MONTEREY CA 93943
1	HARTLEY CONSULTING D HARTLEY 106 WINDSONG LN OAK RIDGE TN 37830
1	F HARTMAN 4850 MARK CTR DR RM 3406 ALEXANDRIA VA 22311
1	THE O'BRYON GROUP J O'BRYON 1608 S TOLLGATE RD BEL AIR MD 21015

NO. OF
COPIES ORGANIZATION

1 SIMIS INC
J GARCIA
200 HIGH ST STE 305
PORTSMOUTH VA 23704

1 OLD DOMINION UNIV
COLLEGE OF ENGRG AND TECHLGY
ENGRG MGMT AND SYS ENGRG
A TOLK
NORFOLK VA 23529

1 USARL
RDRL SL
J SMITH
BLDG 1631 RM 104
WSMR NM 88002-5501

ABERDEEN PROVING GROUND

1 US ARMY TEST AND EVAL CMND
JOINT TEST ELEMENT
P M UGARTE
314 LONGS CORNER RD
APG MD 21005-5055

11 US ARMY MATL SYS ANLYS
ACTVTY
P DEITZ (3 CPS)
P O'NEILL
COL K HAUK
C FOX
B PARIS
J KWON
R MIELE
J THOMAS
G COMSTOCK
392 HOPKINS RD
APG MD 21005-5071

1 US ARMY TEST AND EVAL CMND
ATEC CG
B2202 2ND FL ABERDEEN BLVD
APG MD 21005

1 US ARMY TEST AND EVAL CMND
ATEC ETD
B SIMMONS
B2202 2ND FL ABERDEEN BLVD
APG MD 21005

NO. OF
COPIES ORGANIZATION

1 US ARMY EVALUATION CTR
AEC TD
C WILCOX
B2202 2ND FL ABERDEEN BLVD
APG MD 21005

2 US ARMY EVALUATION CTR
AEC ESD
M DILLEN
D JIMENEZ
B2202 2ND FL ABERDEEN BLVD
APG MD 21005

1 US ARMY EVAL CTR
AEC SVED
R LAUGHMAN
4120 SUSQUEHANNA AVE
APG MD 21005-30113

1 US ARMY MATL SYS ANLYS
ACTVTY
AMXAA GL/REPORTS PROCESSING
392 HOPKINS RD
APG MD 21005-5071

25 DIR USARL
(25 HC) RDRD HRM
P SAVAGE-KNEPSHIELD
RDRL HRM B
C SAMMS
P GRAZAITIS
RDRL HRS C
K MCDOWELL
RDRL SL
D BAYLOR
R FLORES
M STARKS
RDRL SLB
R BOWEN
G KUSINSKI
RDRL SLB A
R DIBELKA
G MANNIX
B WARD (3 CPS)
RDRL SLB D
R GROTE
RDRL SLB E
K AGAN
W LANDIS
M MAHAFFEY

NO. OF
COPIES ORGANIZATION

RDRL SLB G
P MERGLER
RDRL SLB S
R BOWERS
M BURDESHAW
K BURLEY
E HUNT
S SNEAD
RDRL SLE M
B RUTH